DOI: 10.15514/ISPRAS-2019-31(3)-4

An Exploration of Approaches to Instruction Pipeline Implementation for Cycle-Accurate Simulators of «Elbrus»

[1] P.A. Poroshin, ORCID: 0000-0003-0319-5184 <poroshin_p@mcst.ru>
[1,2] A.N. Meshkov, ORCID: 0000-0002-8117-7398 <alex@mcst.ru>
[1] INEUM, 24, Vavilova st., Moscow, 119334, Russia
[2] MCST, 1, Nagatinskaya st., Moscow, 117105, Russia

Abstract. Software simulation is of a big importance during development of processors as they provide access to hardware under development. Cycle-accurate simulators allow software engineers to design and optimize high-performance algorithms and programs with considerations of features and characteristics of processors being in development. This is especially important for architectures, whose performance is mainly achieved by advanced compiler optimizations. One of the core aspects of a cycle-accurate simulator is the way it simulates the pipeline of the target processor. A pipeline model has high impact on an overall structure of a simulator and its potential performance and accuracy. The main goal of this paper is to develop and analyze different approaches to pipeline simulation of "Elbrus" microprocessors, which let us reuse functionality of existing instruction set simulator and achieve good balance of performance and accuracy. We briefly describe features of "Elbrus" microprocessors and specifics of existing instruction set simulator, relevant for cycle-accurate simulation. We make several simple, but general and useful observations about various aspects of pipeline behavior in context of accurate and efficient cycle-accurate simulation of microprocessors. These observations are then used as a basis for justification, development and analysis of the several approaches to the pipeline simulation, described in this paper. We describe four different approaches, starting from simple and obvious one, which is then successively transformed into more advanced ones through several iterations. We analyze limitations of proposed approaches and outline further work.

Keywords: Simulation; Pipeline; Cycle-Accurate Simulator; Microprocessor; Elbrus

For citation: Poroshin P.A., Meshkov A.N. An Exploration of Approaches to Instruction Pipeline Implementation for Cycle-Accurate Simulators of «Elbrus» Microprocessors. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 3, 2019. pp. 47-58. DOI: 10.15514/ISPRAS-2019-31(3)-4

Исследование подходов к реализации конвейера инструкций в рамках потактово-точного симулятора микропроцессоров «Эльбрус»

[1] П.А. Порошин <poroshin_p@mcst.ru>
[1,2] А.Н. Мешков <alex@mcst.ru>
[1] ПАО «ИНЭУМ им. И.С.Брука», 119334, Россия, г. Москва, ул. Вавилова, д. 24
[2] АО «МЦСТ», 117105, Россия, г. Москва, ул. Нагатинская, д. 1, стр.23

Аннотация. Программное моделирование играют важную роль в цикле разработки процессоров, так как они предоставляют доступ к еще не существующему оборудованию. Потактово-точные симуляторы позволяют разработчикам программного обеспечения создавать и оптимизировать программы с учетом особенностей и характеристик разрабатываемых процессоров, что особенно важно для архитектур, которые для достижения высокой производительности в основном опираются на агрессивные

оптимизации компилятора. Одним из ключевых аспектов потактово-точного симулятора является способ моделирования конвейера симулируемого процессора. Программная модель конвейера оказывает большое влияние на общую структуру симулятора и на его производительность и точность. Основной целью данной статьи является разработка и анализ различных подходов к моделированию конвейера микропроцессоров "Эльбрус", которые бы позволяли переиспользовать функционал существующего функционального симулятора без его существенных изменений, и которые бы достигали хорошего баланса производительности и точности. Мы коротко описываем особенности микропроцессоров "Эльбрус" и детали существующего функционального симулятора, важные для потактово-точного моделирования. Мы делаем несколько простых, но достаточно общих и полезных наблюдений о поведении конвейера с позиции точного и эффективного потактово-точного моделирования микропроцессоров. Данные наблюдения используются в качестве основы для обоснования, разработки и анализа нескольких подходов к моделированию конвейера, описанных в данной статье. Всего мы описываем четыре различных подхода, начиная с простого и достаточно очевидного, и заканчивая более сложными, полученными после нескольких итераций совершенствований и усложнений на основе ранее сделанных наблюдений. Для каждого подхода мы анализируем его преимущества, недостатки и фундаментальные ограничения.

Ключевые слова: программное моделирование; конвейер; потактово-точный симулятор; микропроцессор; Эльбрус

Для цитирования: Порошин П.А., Мешков А.Н. Исследование подходов к реализации конвейера инструкций в рамках потактово-точного симулятора микропроцессоров «Эльбрус». Труды ИСП РАН, том 31, вып. 3, 2019 г., стр. 47-58 (на английском языке). DOI: 10.15514/ISPRAS-2019-31(3)-4

1. Introduction

Software based simulation of hardware is a very important tool for development of computing systems. This tool is especially important for software design, as simulators can be used in place of actual still in development (or unavailable for other reasons) hardware. Also simulators can provide wide range of debugging facilities and other information about inner workings of a system being simulated.

One of the widely used classes of simulators is simulators of microprocessors. Different tasks have different needs, so there are simulators with various characteristics. Ones may be oriented at simulation performance; others are aimed at accuracy and precision.

Instruction set simulator (ISS) is a simulator of microprocessor that mostly models a program visible architecture state without considerations of microarchitecture specifics and timings. And while for many tasks this is enough, there is a need for simulators with much greater degree of accuracy that can be used for performance evaluation.

Cycle-accurate simulators (CAS) are such simulators. They are important tools for code efficiency estimation during development of performance critical software and optimizing compilers. Ability to debug performance of code is especially crucial for microprocessor architectures, which achieve high performance not by invisible to programmer microarchitectural features, but mainly by static planning of instruction execution by smart compiler. The «Elbrus» family of microprocessor architectures is such type of architectures.

Modern microprocessors achieve their high performance and clock frequency through use of pipelining. Every cycle-accurate simulator must somehow simulate this pipelining logic to achieve accuracy of its timings. The way a pipeline is represented in a simulator influences various aspects of a simulator, how its components interact and its overall design and characteristics. There are different ways to represent a pipeline and to model it.

In this paper we describe several approaches that were considered as a basis for implementation of the pipeline model during development of the cycle-accurate simulator of microprocessors belonging to the «Elbrus» family of instruction set architectures.

The remainder of this paper has following structure. Section 2 gives brief overview of the «Elbrus» instruction set architecture and describes an existing instruction set simulator used as base for the

cycle-accurate simulator implementation. Section 3 formulates desired properties and requirements for the pipeline model being developed. Section 4 describes in detail several considered approaches to pipeline model organization and explains its discovered advantages and drawbacks. Section 5 gives brief evaluation of described pipeline models. Section 6 is dedicated to other approaches to pipeline simulation that can be found in literature. Section 7 gives concluding remarks and briefly describes plans for further work.

2. Prerequisites

In this section we give some details of the architecture being simulated and of the available instruction set simulator that influence some design decisions around the pipeline model implementation.

2.1 «Elbrus» Family of Instruction Set Architectures

The «Elbrus» family of instruction set architectures is VLIW (Very Long Instruction Word) type of architectures [1]. Performance of this type of architectures is achieved by extracting ILP (Instruction Level Parallelism) through packing in one instruction several sub-operations, which are executed by hardware in parallel. «Elbrus» microprocessors are in-order and have no support of speculative execution (at least in the traditional sense).

In case of the «Elbrus», the packing format is not fixed and there are many ways several sub-operations can be packed in an instruction. Each of these sub-operations can belong to different kinds of operations: arithmetic and logical operations, control flow operations, predicate calculations, memory accesses and so on. And, while generally sub-operations observe only effects of previous instructions, there are several possible interactions of sub-operations within one instruction, for example, in case of a predicated execution.

Another important consideration is the way pipeline stalls work. Firstly, it is worth noting, that in case one sub-operation stalls (for example, because its arguments is not ready yet), the whole instruction stalls, which is a natural result for a VLIW architecture. Secondly, which is more specific for the «Elbrus» architectures, there are a mechanism of prolonged stalls. In simple terms, in some cases (determined by a stall cause and a current pipeline stage) an instruction is not immediately stopped, but effectively after several cycles its results are discarded (as invalid) and it is returned several stages back for its repeated execution in hopes that the original stall will not occur again. This process affects not only the instruction that is not ready for execution, but also several instructions immediately after it. There are two types of such stalls: a 2-cycle one and a 4-cycle one. Moreover, it is possible for several such stalls to interleave, and for such situation there is special pipeline control logic.

Later in this paper we will refer to the pipeline stages of the «Elbrus» microprocessors by following names: F, D, B, R, E0, E1, E2 etc.

2.2 Instruction Set Simulator

Our cycle-accurate simulator was not developed completely from the ground up. An existing instruction set simulator for the «Elbrus» architecture was used as a basis and a starting point for the development of its cycle-accurate version.

This instruction set simulator supports wide range of the various «Elbrus» microprocessors of different architecture iterations via compile time configuration. It also supports both a user mode simulation (with emulation of system calls) and a full system simulation (with MMU logic, peripheral devices etc.). All of this is implemented in a shared code base.

An important feature of the instruction set simulator to consider is how it executes individual (wide) instructions. Execution is divided in two separate steps, conventionally called «read phase» and «write phase». The «read phase» prepares some intermediate data and is mostly side-effect free.

Then the «write phase» uses this intermediate data to complete instruction execution. This way of organization of instruction execution greatly simplifies support of precise exceptions and of some interactions of sub-operations.

3. Requirements to CAS and Its Pipeline Model

There are multiple valid ways to implement a cycle-accurate simulator and its pipeline model, and each design have its trade-offs. Therefore, it is important to define scope and requirements to the cycle-accurate simulator being developed, including its pipeline model implementation. We define following requirements.

- Support of a user mode simulation. At this stage of development it is planned that the cycle-accurate simulator will be used mainly as a tool for debugging performance problems during software and compiler development. For such purposes a user model simulation are used.
- Code reuse with the instruction set simulator. The existing instruction set simulator implements major parts of the «Elbrus» microprocessors, and it would be wasteful to reimplement this functionality separately.
- Configurability. It should be possible to configure the simulator to support the various «Elbrus» microprocessors (like the instruction set simulator) and to enable or disable its different components (for example, for the sake of performance).
- Flexibility. It should be reasonable easy to support new features of next iterations of the «Elbrus» microprocessors. And also, when need arises, it should be possible to adapt the pipeline model for a full system simulation mode.
- Reasonable performance. The cycle-accurate simulator should not be too slow compared to the instruction set simulator. We aim at no more than tenfold slowdown.
- Reasonable accuracy. Of course, exact timing accuracy is not achievable. However, the pipeline model design should not prevent possibility of further accuracy improvement and support of various microarchitectural aspects.

Some of these requirements are conflicting, and we do not expect to simultaneously meet all of them fully, but to achieve some balance between them.

4. Pipeline Simulation of «Elbrus» Microprocessors

In this section we explore several approaches to the pipeline simulation and describe theirs advantages and disadvantages.

4.1 Naïve «Direct Correspondence» Pipeline Model

The first approach that we tried to implement was based on the simple idea of direct and faithful representation of the real pipeline stages in the simulator. These stages would be responsible both for the timing related logic and for the purely algorithmic logic of the corresponding instruction.

We implemented this approach by transforming the «read» and «write» phases of the instruction set simulator into functions representing pipeline stages. During this transformation the «read» and «write» phases were split in parts and the missing pipeline related logic was added to them. To meet the requirement of code reuse, we made code of the new cycle-accurate simulator as base, and implemented the original instruction set simulator by «glueing» stages together into the «read» and «write» phases and removing the pipeline related logic, all of this at compile time and through configuration. Processing of such pipeline model is straightforward.

- Iterate through each pipeline stage.
- For each stage determine which instruction is at this stage and execute functions corresponding to all of the sub-operations of this instruction.

- If there are no stalls - advance instruction to the next stage. Otherwise not advance and propagate stall as necessary. In case of the prolonged stalls simulate related pipeline control logic.

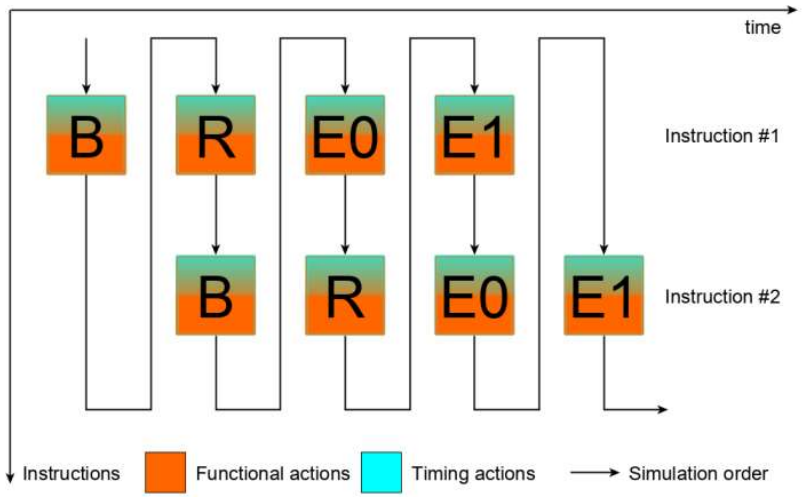

Fig. 1. Simplified illustration of pipeline stage processing in case naïve «direct correspondence» pipeline model

This pipeline model representation should facilitate direct and straightforward support of the various microarchitectural features, as this software model is close to the actual hardware. However, although this idea is conceptually simple, during its implementation we discovered its several major drawbacks.

- Splitting of phases of the instruction set simulator into stages and glueing them back together introduce a major disturbance to the original instruction set simulator functionality. There is no clear way to avoid that. Attempts to fully restore original phases introduce much ad hoc logic, which adds fragility to the whole system. This means there is no easy way to achieve code reuse with this approach.
- In the instruction set simulator there are many unobvious interactions between phases of different sub-operations. These interactions are not easily preserved during splitting of phases.
- While for the most of the operations there is a clear correspondence of phases to pipeline stages, there are exceptions, which add complexity to the glueing process.
- Keeping track of all pipeline stages adds considerable performance overhead, although for most operations only small subset of all pipeline stages are nontrivial (at least in the context of timings).
- Splitting phases into multiple pipeline stage related functions also inhibits compiler optimizations, which impact overall simulator performance.
- After this implementation attempt it became clear that for meeting our code reuse requirement we should minimize changes to the instruction set simulator.

4.2 Smart «Direct Correspondence» Pipeline Model

Next considered approach is a modification of previous one. Its improvements are based on the following key observations.

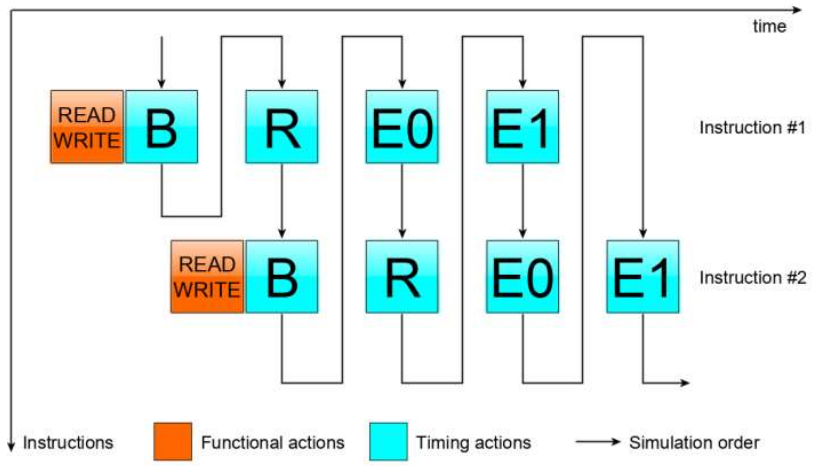

Fig. 2. Simplified illustration of pipeline stage processing in case smart "direct correspondence" pipeline model.

1) Algorithmic behavior of an operation (which is defined by an instruction set architecture and is considered by an instruction set simulator) can influence only an algorithmic behavior of operations of later (or in some cases current) instructions.
2) Algorithmic behavior of an operation determines its timing behavior.
3) Algorithmic behavior of one operation does not directly influence timing behavior of other operation.
4) Timing behavior has no direct influence on an algorithmic behavior (except in some limited number of special cases).
5) Timing behavior of one operation can influence timing behavior of other operation (but usually only in specific ways).
6) Simulator has more information about the execution process than hardware it simulates.
7) Not all details and inner workings of hardware contribute to its timing characteristics.

First six of these observations let us justify the separation of algorithmic and timing logic and moving of the algorithmic logic to the beginning of the instruction processing (right before its pipeline related processing). But we should uphold following conditions

- Algorithmic simulation of the instruction must occur before the algorithmic simulation of the next (in program order) instruction (based on the observation 1).
- Pipeline simulation of the instruction must occur after its algorithmic simulation (based on the observation 2).
- Pipeline simulation of different instructions must occur in order determined by the pipeline state (based on the observation 5).

All of these are satisfied by this approach.

Last observation let us simplify the timing logic by removing all microarchitectural details that are not directly necessary for correctly calculating timing information, as we are interested not in inner workings of hardware, but in timing details.

These transformations should not reduce accuracy of our simulator (except in some rare special cases, which are briefly considered later in this paper).

The algorithm to process such pipeline is very similar to the previous approach. The only difference is that in the beginning of the processing of the first pipeline stage of the instruction we do all algorithmic simulation of this instruction.

This approach let us use functionality of the instruction set simulator (for the algorithmic simulation of instructions) with minimal modifications, which remedy many major drawbacks of the previous approach. But we still have to address the performance concerns, as in this approach the simulator still keeps track of all pipeline stages, even if they are trivial, and the timing logic is still split into multiple independent functions.

4.3 «Fully Speculative» Pipeline Model

The next approach to the pipeline simulation is based on the assumption of stronger the observation 5:

5*) Timing behavior of operation of one instruction can influence timing behavior only of operations of the same or next instructions.

With this modified observation first five observations can be summarized as follows.

- Behavior (algorithmic and timing) of an operation of an instruction cannot depend on the behavior (algorithmic or timing) of operations of next instructions.

This assumption let us simulate all of the instruction's behavior in one go before even considering next instructions. It is just necessary to remember all effects (algorithmic and timing) of the instruction that can influence next instructions. And this is what we do in this approach.

The simulation of pipeline in this approach is as follows:

- Simulate algorithmic behavior of the instruction using the instruction set simulator functionality.
- «Speculatively» simulate timing behavior of the instruction by processing each of its nontrivial stages one by one from first to last, remembering in the process all information about produced effects and their moments in time for use by next instructions (at the same time using such information from previous instructions).
- Move to the next instruction.

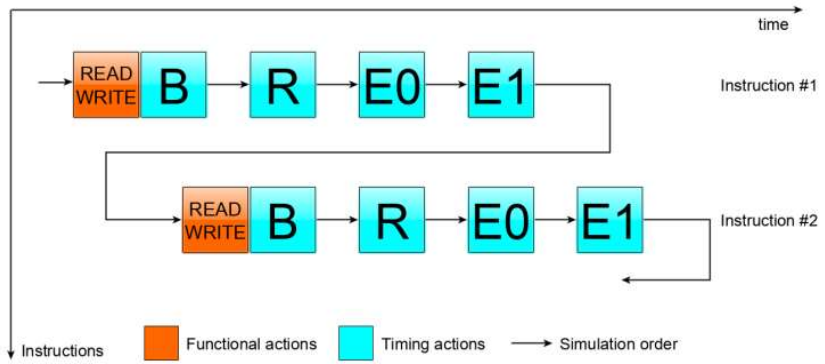

Fig. 3. Simplified illustration of pipeline stage processing in case "fully speculative" pipeline model

Such pipeline organization is expected to be more performant, as it has less overhead related to keeping track of the individual pipeline stages, better processes trivial stages, and in general has more optimization opportunities.

At the same time, with this approach it is necessary to transform the pipeline representation in the new form that supports «speculative» accumulating of effects. This was possible in our case, but may be difficult to achieve in others.

Also, such pipeline model is more complicated and unintuitive. For example, it has no reasonable notion of the current moment in (simulation) time. Time becomes in some sense distributed around the whole pipeline model.

Pipeline is not sole contributor to timing behavior, and it must interact with other components of microprocessor, such as L1 and L2 caches, IB (Instruction Buffer, the component responsible for the fetch of instructions) and others. It may be unfeasible to simulate these components in such «speculative» fashion, and the only reasonable way is the cycle-by-cycle type of simulation. And without a clear «current moment» concept, it is not obvious, when such cycle-by-cycle simulation must occur.

Let us consider L1 cache as a concrete example. Its cycle-by-cycle simulation must occur after all its inputs are available but before its results can influence simulation of the other components (including the pipeline). After careful study of possible interactions of the L1 cache model and the pipeline model we identified that such cycle-by-cycle simulation should occur right after the simulation of the stage R of the instruction. By similar reasoning the cycle-by-cycle simulation of the IB should be placed right after the simulating of the stage F of the instruction. Additional considerations must be made in case of stalls, but overall idea is the same.

Now let us consider interactions between the IB and the L1 cache. In principle, it is possible to the IB request of the future instruction to interfere with the L1 cache state observed by the current instruction. Therefore, it is possible to the timing behavior of the future instruction to influence the timing behavior of the current instruction, which is a violation of our earlier assumption. It means that in this approach we cannot accurately simulate some interactions between various microprocessor components.

Another example of violation of our assumption is the complex interactions during the interleaving of prolonged stalls, where stall of the next instruction can influence stall latency of the current instruction.

Overall, while this approach promises performance improvement, it sacrifices accuracy and flexibility and introduces additional complexity.

4.4 «Hybrid» Pipeline Model

The last approach to the pipeline simulation considered in this paper is a combination of second and third approaches. This pipeline model tries to retain accuracy of the smart «direct correspondence» model and to achieve some of the performance benefits of the «fully speculative» model. It is based on the two additional observations:

8) Pipeline behavior of an instruction interacts with pipeline behaviors of other instructions and other components at specific pipeline stages.
9) There are continuous sequences of stages that executed uninterrupted (without stalls and influence from other instructions and components).

For example, after the stage E2 there is no possibility of any stall and all further timing behavior of the instruction is predetermined. So it is possible to simulate such continuous uninterrupted sequences of stages speculatively in a manner similar to the «fully speculative» approach, but without the risk of decreasing timing accuracy. And after the instruction reached the pipeline stage E3, we can stop keeping track of it, as its timing behavior is completely simulated (partly normally and partly speculatively) at this point. This significantly decreases the pipeline simulation performance overhead and the overhead of dealing with trivial stages.

Processing of such pipeline model is very similar to the smart «direct correspondence» approach:

- Iterate through each pipeline stage.

- For each stage determine which instruction is at this stage.
- If it is a new instruction, then simulate its algorithmic behavior.
- If it is the first stage of an uninterrupted sequence, then speculatively simulate all stages of this sequence.
- If there are no stalls, then advance the instruction to the next stage. Otherwise not advance and propagate stall as necessary. In case of prolonged stalls simulate related pipeline control logic.

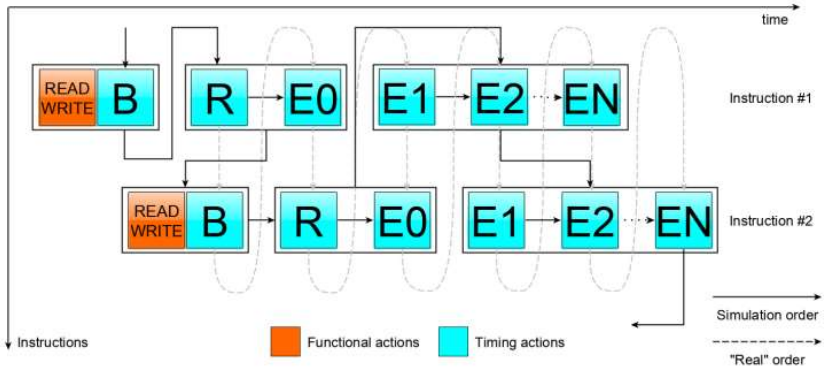

Fig. 4. Simplified illustration of pipeline stage processing in case "hybrid" pipeline model

Overall, this approach let us partially get performance gain of the «fully speculative» approach without its major drawbacks of sacrificing accuracy.

Unfortunately, all described approaches (except the naive one) do not cover the special case of the timing behavior influencing the algorithmic behavior. Example of such situation is operations that generate a predicate based on readiness of its arguments. Researching of ways to address this is part of our future work, and we hope it will be possible to implement a solution within the «hybrid» approach.

5. Evaluation

Although the cycle-accurate simulator is still in development and there is work to be done (for example, memory subsystems are not fully implemented yet and are greatly simplified), it is worth to do some preliminary evaluation of the pipeline model implementations described in this report.

Here we will consider only the «fully speculative» and the «hybrid» models, as the «direct correspondence» models were abandoned much earlier in the development and it is hard to make a fair comparison of them to the other models.

We compare the relative performance and the total number of the simulation cycles that were needed for the test completion. The instruction set simulator is used as a baseline. Individual test cases consist of the executing on the simulator part of one of the SPEC CPU95 benchmarks. Results presented in Table 1.

At this stage of the development we do not have a reasonable cycle count reference that we can use, because, for example, our simulators do not do proper simulation of various memory accesses. Nevertheless, we hope to get rough estimate of contribution of the more detailed simulation of the pipeline by the "hybrid" model to the total cycle count.

Results show that on average the «hybrid» model is slower than the «fully speculative» model by ~20%. At the same time, average difference in total cycle count is around 0.5% with one significant outlier «146.wave5» with the cycle count difference of 6.1%. We expect that this is because less accurate simulation of the prolonged stalls in the «fully speculative» pipeline model.

It is possible to optimize both models and the performance difference after optimizations can change, but we expect that the «hybrid» model will always be slower. Despite this overall we consider the «hybrid» model as a better approach as it is more fully meets our requirements of accuracy and flexibility, and in a need of performance it should be possible to configure the «hybrid» model accordingly.

Table 1. Performance and total cycle count relative to instruction set simulator.

Test	«Hybrid» CAS		«Fully speculative» CAS	
	Relative Performance	*Relative cycle count*	*Relative Performance*	*Relative cycle count*
099.go	0,192	1,747	0,218	1,747
101.tomcatv	0,271	1,415	0,323	1,424
102.swim	0,329	1,983	0,407	1,981
103.su2cor	0,277	1,415	0,322	1,420
110.applu	0,218	1,999	0,266	2,001
124.m88ksim	0,243	1,151	0,299	1,151
126.gcc	0,291	1,376	0,341	1,378
129.compress	0,222	1,522	0,286	1,539
130.li	0,195	2,102	0,224	2,104
132.ijpeg	0,218	1,738	0,261	1,749
134.perl	0,252	1,541	0,301	1,553
141.apsi	0,248	2,364	0,286	2,379
146.wave5	0,328	1,734	0,398	1,840
147.vortex	0,250	1,600	0,308	1,601

6. Related Work

Cycle-accurate simulation of modern microprocessors is a very active area of research. But only small portion of this research is focused on simulating of general purpose VLIW microprocessors, let alone on the «Elbrus» architecture. And many of the available approaches do not quite translate to the «Elbrus» specifics.

Approaches of simulating a pipeline of VLIW microprocessors, similar to the «direct correspondence» approaches, are described in [2-4]. However, they do not address the issue of code reuse in the presence of an instruction set simulator.

All of the approaches described in this paper are execution-driven. Trace-driven simulation is one of the alternatives [5-9]. The basic idea of the trace-driven approach is a separation of the whole simulation process in two phases: generation of some data (trace), that represents an execution path, and using that data as an input for a cycle-accurate simulation of some microprocessor aspect. Trace can be generated by real hardware or other simulator (for example, an instruction set simulator). This approach gives benefits, similar to ones we aim to achieve by separation of algorithmic logic and timing logic introduced in our second approach, but makes extremely difficult to account for a possible dependence of an algorithmic behavior on a timing behavior (which we are planning to address in future work in our approach), as these interactions cannot be captured in trace during its generation before cycle-accurate simulation.

The pipeline representation, similar to our «fully speculative» approach, is used in [10]. Authors describe in details various aspects of the pipeline simulation (occupancy of stages, operand

dependencies and control flow considerations), but do not discuss limits of this approach and complexities of interaction of such pipeline model with other components of microprocessor.

7. Conclusions and Future Work

Software based simulation of microprocessors is a very important tool. There are many possible ways to implement such simulators, each of them with its own set of advantages and disadvantages.

In this paper we explored several approaches to the pipeline simulation in the context of the cycle-accurate simulation of the «Elbrus» microprocessors. We made several simple, but general and powerful observations, which were used as the foundation for the design of the various pipeline models and for the analysis of their advantages and drawbacks. We described several of such approaches that were considered and at least partially implemented during development of our cycle-accurate simulator.

The cycle-accurate simulator described in this paper is still in active development. In the future work we are planning to address the issue of dependence of the algorithmic behavior of the instruction on the timing behavior and to explore additional ways to optimize performance of the simulation.

References / Список литературы

[1]. Kim A.K., Perekatov V.I., Ermakov S.G. Microprocessors and computing complexes of the Elbrus family. Piter, 2013, 272 p. (in Russian). / Ким А.К., Перекатов В.И., Ермаков С.Г. Микропроцессоры и вычислительные комплексы семейства «Эльбрус». Питер, 2013, 272 стр.

[2]. Cuppu Vinodh. Cycle accurate simulator for TMS320C62x, 8 way VLIW DSP processor. University of Maryland, College Park (1999).

[3]. Barbieri I. et al. Flexibility, Speed and Accuracy in VLIW Architectures Simulation and Modeling. In Proc. of the 2002 WSEAS International Conference on Electronics and Hardware Systems, 2002, pp. 1661-1665.

[4]. Barbieri I., Bariani M., Raggio M. A VLIW architecture simulator innovative approach for HW-SW co-design. In Proc. of the IEEE International Conference on Multimedia and Expo, 2000, vol. 3, pp. 1375-1378.

[5]. Uhlig R. A., Mudge T. N. Trace-driven memory simulation: A survey. ACM Computing Surveys, vol. 29, №. 2, 1997, pp. 128-170.

[6]. Joshua J. Y. et al. The future of simulation: A field of dreams. Computer, vol. 39, №. 11, 2006, pp. 22-29.

[7]. Agarwal A., Huffman M. Blocking: Exploiting spatial locality for trace compaction. ACM SIGMETRICS Performance Evaluation Review, vol. 18, №. 1, 1990. pp. 48-57.

[8]. Cho S. et al. TPTS: A novel framework for very fast manycore processor architecture simulation. In Proc. of the 2008 37th International Conference on Parallel Processing, 2008, pp. 446-453.

[9]. Lee H. et al. Two-phase trace-driven simulation (TPTS): a fast multicore processor architecture simulation approach. Software: Practice and Experience, vol. 40, №. 3, pp. 239-258.

[10]. Böhm I., Franke B., Topham N. Cycle-accurate performance modelling in an ultra-fast just-in-time dynamic binary translation instruction set simulator. In Proc. of the 2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2010, pp. 1-10.

Информация об авторах / Information about authors

Павел Александрович ПОРОШИН получил степень магистра в 2017 году в Московском физико-техническом институте. В настоящее время работает в ИНЭУМ им. И.С. Брука в качестве инженера-программиста. В область его научных интересов входят программное моделирование вычислительных систем и потактово-точное моделирование микропроцессоров.

Pavel Alexandrovitch POROSHIN received his MS degree from Moscow Institute of Physics and Technology in 2017. He is currently working as software engineer at INEUM. His research interests include software simulation of computing systems and cycle-accurate simulation of microprocessors.

Алексей Николаевич МЕШКОВ получил степень кандидата технических наук в ИНЭУМ им. И.С. Брука в 2013 году. В настоящее время является начальником отдела в АО «МЦСТ». Область научных интересов включает программное моделирование и верификацию компьютерных систем.

Alexey Nikolaevitch MESHKOV received his PhD degree at INEUM in 2013. He is currently working as a chief of department at MCST. His research interests include software modelling and verification of computer systems.