Article

Top-Down models across CPU architectures: Applicability and comparison in an HPC environment

Fabio Banchelli * , Marta Garcia-Gasulla , Filippo Mantovani *

Posted Date: 10 August 2023

doi: 10.20944/preprints202308.0803.v1

Keywords: Performance models; Top-Down model; HPC applications; MareNostrum 4; A64FX; Power 9; Zen 2

Article

Top-Down Models across CPU Architectures: Applicability and Comparison in an HPC Environment

Fabio Banchelli [1,*,†], Marta Garcia-Gasulla [2,†] and Filippo Mantovani [3,†]

[1] fabio.banchelli@bsc.es
[2] marta.garcia@bsc.es
[3] filippo.mantovani@bsc.es
[*] Correspondence: fabio.banchelli@bsc.es
[†] Barcelona Supercomputing Center Plaça Eusebi Güell, 1-3 08034 Barcelona (Spain).

Abstract: Top-down models are defined by hardware architects to provide information on the utilization of the different hardware components. The target is to isolate the users from the complexity of the hardware architecture while giving them insight into how efficiently the code is using the resources. In this paper, we explore the applicability of 4 top-down models defined for different hardware architectures powering state-of-the-art HPC clusters (Intel Skylake, Fujitsu A64FX, IBM Power9, and Huawei Kunpeng 920) and propose a model for AMD Zen 2. We study a parallel CFD code used for scientific production to compare these 5 Top-Down models. We evaluate the level of insight achieved, the clarity of the information, the ease of use, and the conclusions that each one allows us to reach.

Keywords: performance models; top-down model; HPC Applications; MareNostrum 4; A64FX; Power 9; Zen 2

1. Introduction and related work

Diversity in CPU architectures is the new reality in the HPC industry. The five top spots in the Top500 list of November 2022 [1] include three different CPU architectures: x86, Arm, and IBM POWER. To increase the diversity of the panorama, each of these CPU architectures has different implementations by the vendors (e.g., x86 by Intel and AMD, Arm by Fujitsu, Nvidia and Huawei). With such diversity, it becomes increasingly difficult to establish cross-platform methods to evaluate the efficient use of these hardware systems. The complexity may not arise with benchmarks such as HPL or HPCG, but with production scientific applications.

There are multiple performance analysis models that try to *i)* measure performance, and *ii)* identify performance bottlenecks. Furthermore, some models are also able to give hints on how to circumvent said bottlenecks.

For example, the Roofline model [2,3] plots performance in relation to arithmetic intensity (or operational intensity). This model gives feedback in whether a certain code is compute bound or memory bound and also tells how far is the execution from the theoretical peak. Depending on the level of analysis, the model can define different roofs (e.g., accounting for cache levels and main memory) [4]. The arithmetic intensity of a code can be either defined *i)* theoretically, by counting the number of logical operations that the algorithm requires, or *ii)* empirically, by measuring the operations during execution. The second method yields different results because it includes the modifications that the compiler might introduce on the implementation of the algorithm [5]. For codes which are far from the theoretical peak, the main limitation of the Roofline model is to tell where is the performance lost. For memory bound codes, the performance will *eventually* be bounded by the memory subsystem. But that does not mean that the current implementation or execution shares the same bottleneck.

The question is *Where are the execution cycles being lost?* The Top-Down model tries to answer this question.

1.1. Top-Down model

By counting and classifying the execution cycles, the Top-Down model points to the current limiting factor of the application. This model was defined by Intel originally described by Intel [6] and later established as the Intel TMAM methodology [7]. The model is structured as a tree of metrics that organize cycles depending on what were they used for (e.g., compute resources, memory resources, lost due to stalls, etc.) The proposed evaluation methodology is to drill down the path where most cycles are lost. For Intel CPUs, there is an official definition for each level of the hierarchy. The deeper the hierarchy goes, the more micro-architecture specific it becomes. This might make it difficult to compare results between CPUs.

In contrast to Intel CPUs, there is no official Top-Down model defined by AMD. There have been efforts in the past that try to map the model from Intel into the AMD 15h, Opteron and R-Series Family processors [8]. The work highlights the limitations of the mapping due to differences in micro-architecture as well as the available hardware counters. Although not labeled *Top-Down*, there are also tree-like hierarchies for other architectures. These have been defined by the vendor and may not match with the definitions of TMAM.

In this work, we leverage the past experience with a production CFD code named Alya [9] to explore the insight that the Top-Down model can provide. We also analyze the applicability of the models in different CPU architectures and if the model hierarchies are comparable.

1.2. Contributions

1. Define the Top-Down model in AMD Zen 2
2. Implement the Top-Down model in Intel Skylake, AMD Zen 2, A64FX, Power9, and Kunpeng 920 CPUs
3. Apply the Top-Down model to study the effect of code modifications in a production HPC code across different CPU architectures
4. Compare of Top-Down model across systems with different CPU architectures

2. HPC Systems and their Top-Down model

In this section, we introduce the hardware under study. We explain how the Top-Down model of each machine was constructed and how to interpret the most relevant metrics. We use the Top-Down model of TMAM as a baseline to compare with other machines. Please refer to Appendix A for a detailed listing on how to compute each metric.

In Table 1 we summarize the cluster configurations of all the HPC systems under study. We include some relevant hardware features as well as the system software stack. We also show a general summary of the Top-Down model on each machine.

Table 1. Hardware and software configurations for MareNostrum 4, CTE-AMD, CTE-Arm, CTE-Power, and CTE-Kunpeng

	MareNostrum 4	CTE-AMD	CTE-Arm	CTE-Power	CTE-Kunpeng
Cluster architecture					
Number of nodes	3456	33	192	52	16
CPU Model	Xeon Platinum 8160	EPYC 7742	FX1000	Power9 8335-GTH	Kunpeng 920
Architecture	x86_64	x86_64	aarch64	ppc64le	aarch64
CPUs per node	2	1	1	2	2
Cores per CPU	24	64	48	20	64
Frequency [MHz]	2100	2250	2200	3000	2600
Floating-point performance					
Vector/SIMD extension	AVX512	AVX2	SVE / NEON	VSX	NEON
Vector/SIMD size [B]	8	4	8 / 2	2	2
Peak performance [GFlop/s]	67.20	54.16	70.40 / 17.60	24.0	10.40
Memory subsystem					
L1 Cache [KiB]	32 private	32 private	64 private	32 private	64 private
L2 Cache [MiB]	1 private	0.5 shared	32 shared	0.5 shared	0.5 private
L3 Cache [MiB]	33 shared	16 shared	-	1 shared	32 shared
Main Memory [GB]	96	1024	32	512	256
System software					
Kernel	Linux/4.4.120	Linux/4.18.0	Linux/4.18.0	Linux/4.14.0	Linux/4.14.0
OS	SUSE/12.2	Rocky Linux/8.5	Red Hat/8.1	Red Hat/7.5	CentOS/7
Compiler	intel/2020.1	intel/2018.4	arm/20.3	pgi/20.4	gcc/11.2.0
PAPI Library	papi/6.0.0	papi/6.0.0.1	papi/git-2020-10-08	papi/6.0.0	papi/6.0.0.1
MPI Library	impi/2018.4	impi/2018.4	openmpi/4.0.5	openmpi/3.0.0	openmpi/4.1.3
Tracing Library	extrae/3.8.3	extrae/3.8.3	extrae/3.8.3	extrae/3.8.3	extrae/3.8.3
Top-Down model					
Hierarchy levels	2	1	3	6	3
Metrics relative to	Total *Slots*	Total *Slots*	Parent metric	Parent metric	Total *Slots*[1]
Parameters	*Pipeline_Width*	*Pipeline_Width*, *Mispredict_Cost*	-	-	*Pipeline_Width*
Required event sets	2	2	3	10	2

2.1. MareNostrum 4 general purpose

MareNostrum 4 is flagship Tier-0 supercomputer hosted at BSC. The general purpose partition (from here on, simply MareNostrum 4) has 3456 nodes housing two Intel Xeon Platinum 8160 CPUs. This partition ranked 29th in the Top500 of June 2019.

The Top-Down model in MareNostrum 4 has been constructed following Intel's TMAM definition. Figure 1 shows a schematic view of the model. This version of the model, focuses on the *Slot* occupation at the boundary between the front-end and the back-end of the processor pipeline. One slot can be either consumed by one micro-operation (μOp) coming from the front-end or get lost because the corresponding resource in the back-end is busy. In the case of the Skylake CPU, there are four slots available per cycle (up to four μOps can be dispatched per cycle). These slots can fall under one of the following categories:

- **Frontend Bound** the slot was lost due to not having enough μOPs to execute.
- **Bad Speculation** the slot was used, but to execute a speculative instruction that was later cleared.
- **Backend Bound** the slot was lost due to the back-end resources being occupied by an older μOP.
- **Retiring** the slot was used for an instruction that eventually retired. A high number in this metric means that the pipeline has not lost slots due to stalls. This does not mean that the hardware resources are being utilized efficiently, it only means that work is flowing into the pipeline.

Each category is further divided into more detailed metrics. For example, the *Backend Bound* category is divided into *Memory Bound* and *Core Bound*. In this version of the Top-Down model, child metrics add up to their parent's value. All values represent the portion of total slots of the execution. The sum of all metrics in the first level adds up to one. With the hardware counters available in MareNostrum 4, we were able to construct two levels of the Top-Down model for the Skylake CPU. These two levels are sufficiently generic to be compared to other CPUs, even if based on different architectures.

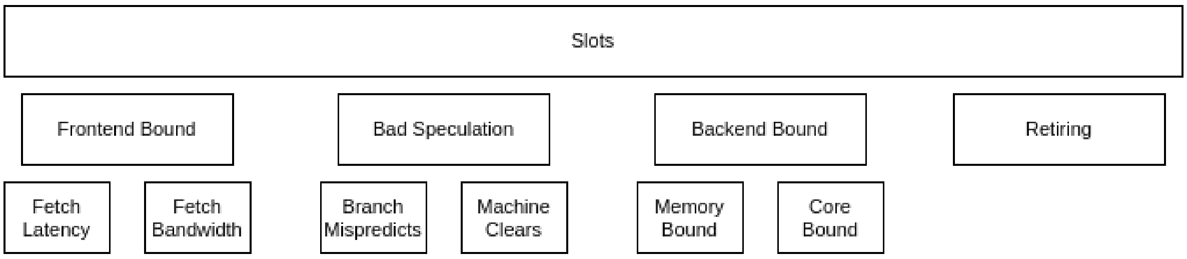

Figure 1. Top-Down model hierarchy in MareNostrum 4

2.2. CTE-AMD

CTE-AMD is part of the *CTE* clusters deployed at BSC. It has 33 nodes made up of AMD Rome processors, similar to the CPUs of the Frontier supercomputer that was installed in 2021 at ORNL and ranked 1st in the Top500 of June 2022. Our work extends the previousy defined mapping of the R-Series Family processors to the EPYC 7742 CPU hosted in the CTE-AMD. Due to the limitations on the system software (i.e., hardware counters and PAPI library), we were only able to map the first level of metrics. Figure 2 shows a schematic view of the model.

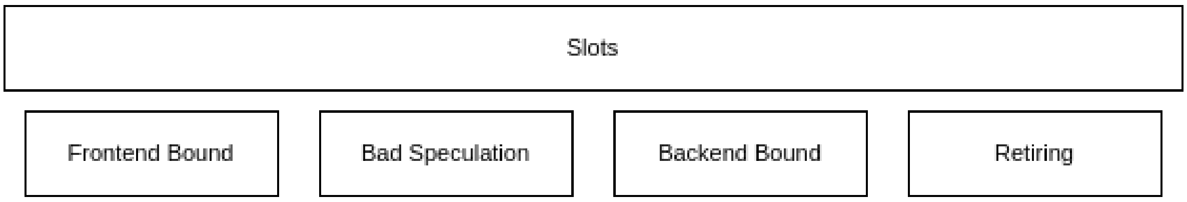

Figure 2. Top-Down model hierarchy in CTE-AMD

The Top-Down model in CTE-AMD shares the first level of metrics with Intel's model. Howevere, despite having sharing the same name, there are some differences between the metrics in Intel and AMD, mainly:

- **Frontend Bound** is based on the counter `UOPS_QUEUE_EMPTY`, which does not take into account if the back-end is stalled or not.
- **Bad Speculation** requires a micro-architectural parameter, *Mispredict_Cost*, which represents the average cycles lost due to a misprediction. We define this constant as 18 based on the publicly available experimental data [10].

Like with MareNostrum 4, the metrics of each level are between zero and one, and represent a portion of slots of the total execution.

2.3. CTE-Arm

CTE-Arm is another cluster of the *CTE* systems. It is powered by the A64FX chip developed by Fujitsu and based on the Arm-v8 instruction set. The architecture of the cluster is the same as the one of Fugaku supercomputer, which ranked 1st in the Top500 of June 2020.

The Top-Down model in CTE-Arm has been constructed based on the official micro-architecture manual published by Fujitsu [11]. The manual uses the term *Cycle Accounting* instead of *Top-Down model*. Figure 3 shows a schematic view of the model. The model is presented as a tree of hardware counters instead of metrics. It does not require any micro-architectural parameter.

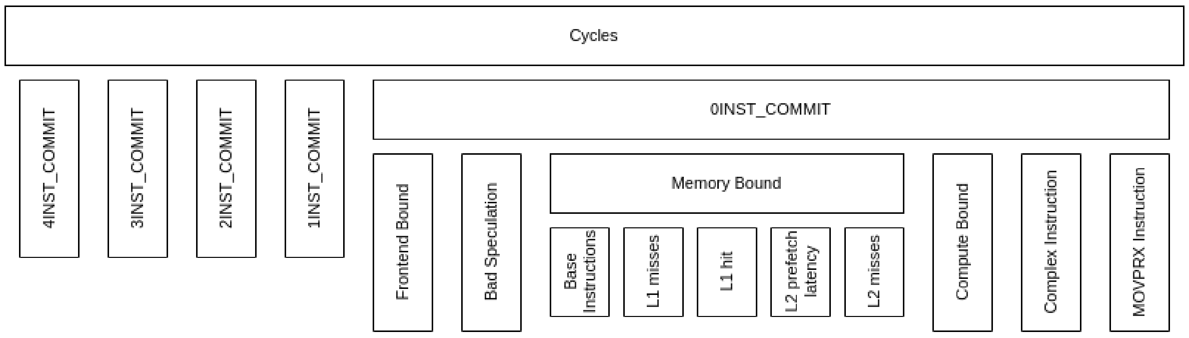

Figure 3. Top-Down model hierarchy in CTE-Arm

The hierarchy is up to five levels deep. Almost all nodes in the hierarchy correspond to a hardware counter, and each node is the sum of all its children nodes. This means that there is some redundancy between counters of the hierarchy, but it also means that it is not necessary to construct the whole hierarchy in order to study the first and second levels. There are some metrics marked as *Other* which correspond to the difference between the parent node and the aggregation of all children nodes.

In contrast to the Top-Down model in MareNostrum 4 and CTE-AMD, the metrics in CTE-Arm focus on the cycles lost during the commit stage of the instructions (not in the boundary between front-end and back-end). The first level of the hierarchy classifies cycles depending on how many instructions were committed. The best case scenario is four instructions, while the worst is no instructions committed. Since the situation of zero commits in a cycle is the most critical, the Top-Down model hierarchy focuses on this path.

In contrast to MareNostrum 4 and CTE-AMD, the metrics in CTE-Arm are relative to the parent. This means that the metric represents a portion of the cycles of the parent metric, instead of the total execution.

2.4. CTE-Power

CTE-Power is part of the *CTE* clusters and spans across 52 nodes housing two IBM Power9 8335-GTH CPUs. Summit and Sierra are two supercomputers based on the same CPU as CTE-Power that ranked 1st and 3rd in the Top500 of June 2018.

The Top-Down model in CTE-Power has been constructed based on the official PMU user guide published by IBM [12]. Figure 4 shows a schematic view of the model. It does not require any micro-architectural parameter. The manual includes a tree diagram of the first two levels of the model hierarchy. Lower levels are only referenced using their respective hardware counters.

- **ICT empty** no instruction to complete (the pipeline is empty). Similar to *Frontend Bound* in other models.
- **Issue hold** next-to-complete instruction (i.e., oldest in the pipeline) is held in the issue stage.
- **Pipeline stall** similar to the *Backend Bound* category in previous models.
- **Thread blocked** next-to-complete instruction is held because an instruction from another hardware thread is occuping the pipeline.
- **Instruction latency** cycles waiting for the instruction to finish due to the pipeline latency.
- **Completion cycles** cycles in which at least one instruction was completed. Similar to *Retiring* category in previous models.

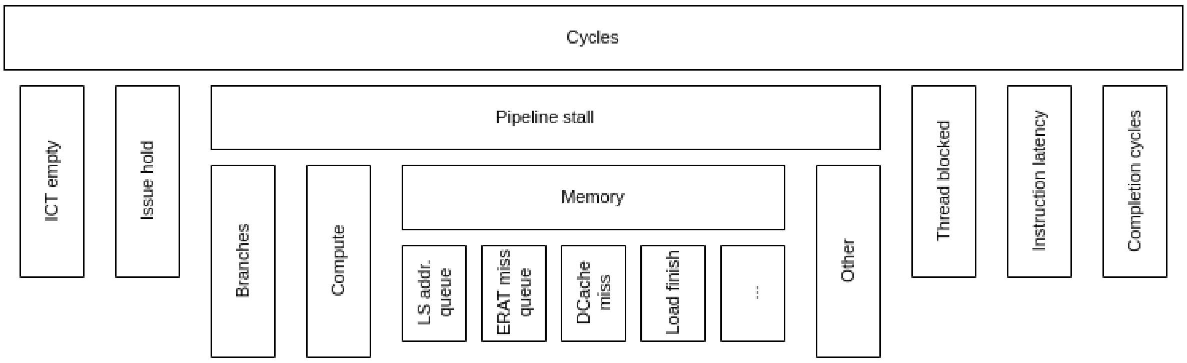

Figure 4. Top-Down model hierarchy in CTE-Power

The hierarchy is up to six levels deep. Starting from the second level, some nodes of the hierarchy do not represent an actual hardware counter that is available in the machine, but the aggregation of the counters below it. This means that the Top-Down model in CTE-Power requires to construct the whole tree if the study needs to go beyond the first level of the hierarchy. In this work, we only reach up two the third level and only following the *Memory* path, since it is the most relevant for the application under study. Like with CTE-Arm, the metrics in CTE-Power represent always a portion of the parent's cycles and not the whole execution.

From the official documentation, it is unclear whether the categories of the model are disjoint or not. Furthermore, the *Instruction latency* category (labeled as *Finish-to-completion* in the manual) is mentioned once when listing the top level of the model, but not mentioned later on. Without more details, it is not possible to drill down the path of *Instruction latency* if it were to be the main limiting factor of the application under study.

In MareNostrum 4, the *Retiring* category branches into other metrics that take into account the pipeline latency (we do not include this branch in our work because we are limited by the available counters). One could argue that the *Instruction latency* in CTE-Power could be included under *Completion cycles* similar to the model in MareNostrum 4.

2.5. CTE-Kunpeng

CTE-Kunpeng is a cluster powered by the Arm-based Kunpeng 920 CPU which was deployed in 2021 at BSC as a result of a collaboration between BSC and Huawei. The model in CTE-Kunpeng aims to mimic the same structure and naming convention as in MareNostrum 4. The name of the hardware counters are different than in the Skylake CPU, and it is unclear whether they cover the same situation exactly. Nonetheless, the formulation of the metrics emulates the original definition by Intel. The model in CTE-Kunpeng defines three levels. Figure 5 shows a schematic view of the model.

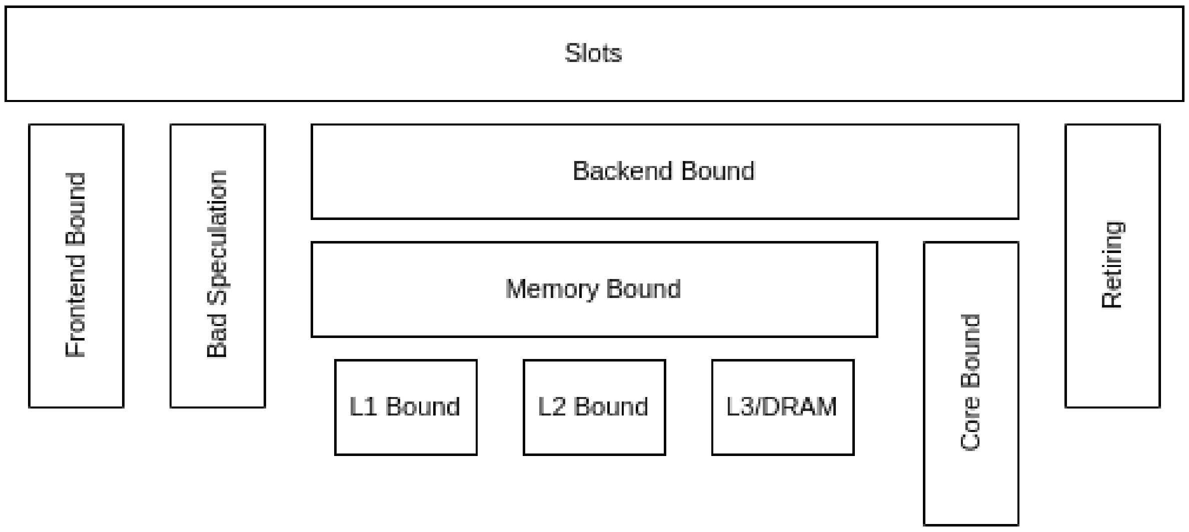

Figure 5. Top-Down model hierarchy in CTE-Kunpeng

The first level of the hierarchy includes metrics which represent a portion of the total cycles. On the other hand, the *Memoy Bound* branch in the second and third level defines metrics which are relative to the cycles in which the core was stalled waiting for memory. CTE-Kunpeng is the only case we cover in this document where metrics are relative to different quantities depending on the level of the hierarchy. This makes it more difficult to compare metrics between models.

3. HPC Application: Alya

Alya is a Computational Fluid Dynamics code developed at the Barcelona Supercomputing Center and it is part of the PRACE Unified European Applications Benchmark Suite[13]. The application is written in Fortran and parallelized with the MPI programming model. In this work, we run a version and input of Alya that we previously studied [9]. This version implements a compile time parameter `VECTOR_SIZE` which changes the packing of mesh elements to expose more data parallelism to the compiler. In our previous study, we explored how the execution time, executed instructions, and IPC, evolved while increasing `VECTOR_SIZE`. We measured that the elapsed time curve has a *U* shape, with `VECTOR_SIZE` 32 being the best configuration. Our study was limited to MareNostrum 4 and compared different compilers. In this work, we leverage our previous knowledge of Alya to run in multiple clusters and construct the Top-Down model for each one of them.

3.1. General structure

Our use case is the simulation of a chemical combustion, thus it includes computing the chemical reaction, the temperature of the reaction and the velocity of the fluid.

For our particular input, the execution of Alya is divided into five integration steps (or timesteps). Each step is further divided into phases. Figure 6 shows a timeline of one timestep in MareNostrum 4. The x-axis represents time, while the y-axis represents MPI processes. The timeline is color-coded to show the different execution phases. In this work, we explore the Nastin matrix assembly phase, which represents the longest time in a timestep (blue regions of the timeline) and corresponds to the non-compressible fluid.

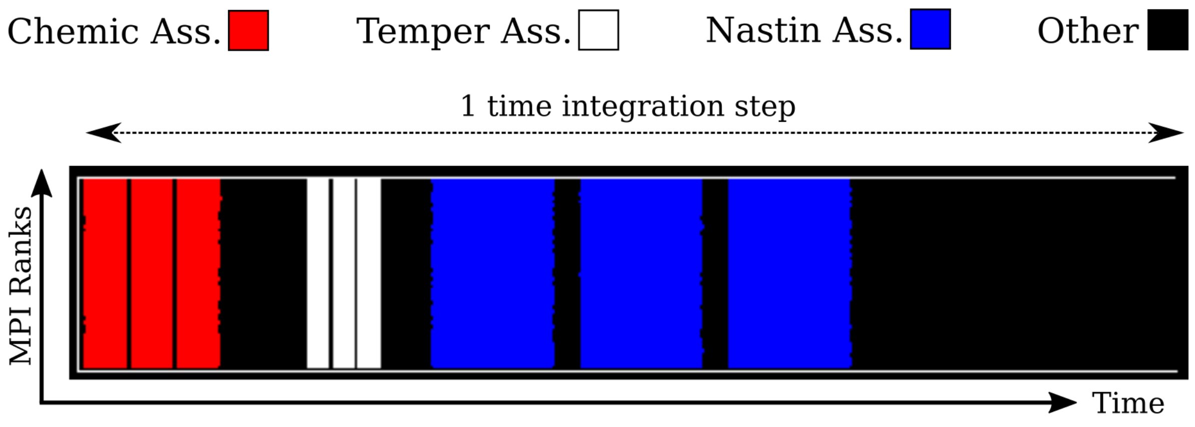

Figure 6. Timeline of one timestep in Alya

4. Model implementation

4.1. Tools

PAPI [14] is a library that leverages the portability of `perf` and has an easy programming interface. It also defines a list of generic counters called *presets* that should be available on most CPUs. However, even with the same name, PAPI counters may not measure the same event when read in different systems. For example, `PAPI_VEC_INS` may read all issued vector instructions on a CPU while only measuring issued arithmetic vector instructions on another system.

Extrae is a tracing tool which intercepts MPI calls and other events during the execution of an application and collects runtime information. The information gathered includes *i)* performance counters via PAPI, *ii)* which MPI primitive was called, and *iii)* which processes were involved during the communication. All this information is stored into a file called a trace. Each record in a trace file has an associated timestamp. Tracing MPI applications with Extrae does not require to recompile the application.

Paraver [15] is a visualization tool that helps navigate the traces generated by Extrae. A common visualization mode when using Paraver is the timeline. Timelines represent the evolution of a given metric across time. Figure 7 shows two examples of timelines. The x-axis represents time and the y-axis represents the MPI processes. Each burst is color coded. For quantitative values (top), the color scale goes from dark blue (high values) to light green (low values). For qualitative values (bottom), each color represents a different concept (e.g., MPI primitive).

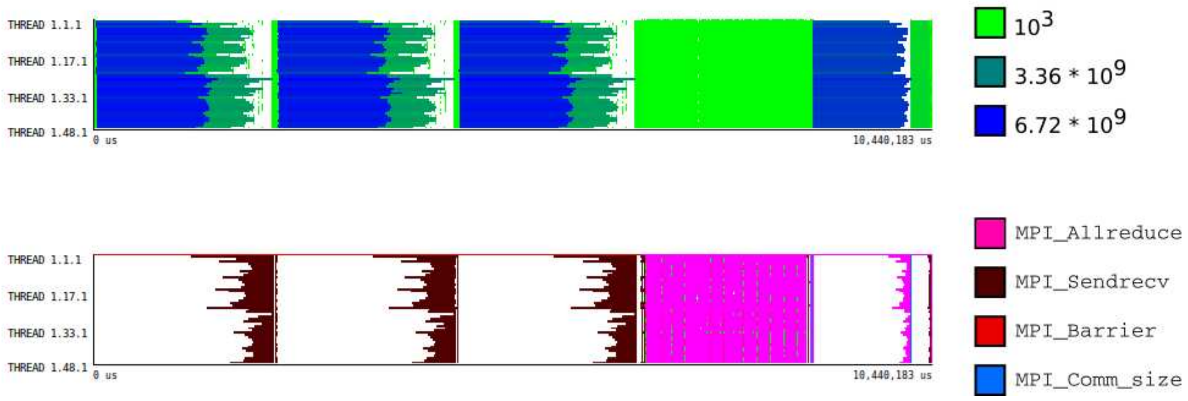

Figure 7. Paraver timeline examples. Top: quantitative representation of number of instructions. Bottom: MPI primitive calls.

Daltabaix is a data processing tool we implemented that takes Paraver traces and computes the metrics of the Top-Down model for a given CPU. The list of metrics, which counters are necessary and how to combine them is stored in a configuration file for each supported CPU model. Since the number of hardware counters that can be polled with a single PAPI event set is not enough to construct the whole Top-Down model in each machine, Daltabaix collects counters from traces of executions with different event sets. For parallel executions, Daltabaix takes the sum of all measurements across ranks to compute the metrics of the Top-Down model. The reason for this method is that we define a *budget* of *Clocks* or *Slots* (depending on the system) that the application has consumed regardless of which cycle belongs to which core.

4.2. Measurement methodology

In each machine, we run Alya using one full node mapping one MPI rank to each core. We configured the simulations to run with five timesteps. We manually instrumented the code to perform measurements at the beginning and end of the Nastin matrix assembly phase in each timestep. We verified that the variability of the hardware counters is under 5% between runs, so we assume that we can operate between counters that belong to different executions.

Figure 8 shows a schematic view of the workflow of our study. From left to right: *i)* we execute Alya with Extrae instrumentation. *ii)* Extrae calls the PAPI interface which will poll the hardware counters. *iii)* At the end of the execution, Extrae generates a Paraver trace which is fed into Daltabaix. *iv)* Following the definition of the configuration file, Daltabaix extracts the relevant metrics for the Top-Down model calling the Paraver command line tool. Metrics are printed out in tabular form and also stored into a separate file.

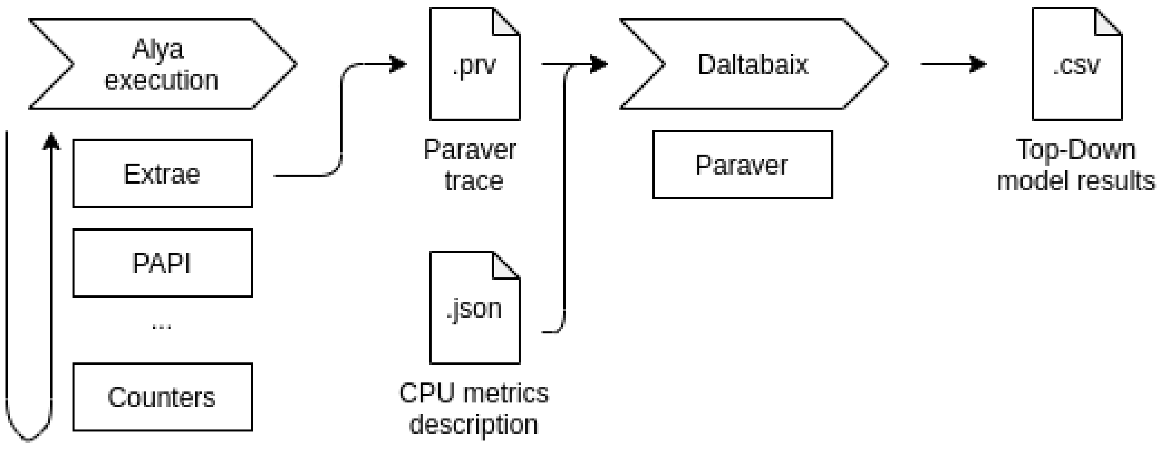

Figure 8. Workflow from application execution to model results

5. Results

In this section we present the Top-Down model results for Alya in each system under study. We lay down the model hierarchy as tables. Rows represent metrics while each column represents runs with a given `VECTOR_SIZE`. The metrics above the dotted line are not part of the Top-Down model but provide complementary information that we consider relevant to analyze the metrics as a whole. Each cell contains the value of a given metric for a given run, and is color-coded with a gradient from red to (metric equals zero) to green (metric equals one). The reader should keep in mind that depending on the machine, the metrics represent a portion of the total amount of consumed slots (or cycles), while on others they represent a portion of the parent metric.

5.1. MareNostrum 4

Table 2 shows the Top-Down model for Alya in MareNostrum 4. We observe that the cycles decrease up to a `VECTOR_SIZE` of 32, after which they start to bounce back up. The number of executed instructions and the IPC also coincide with our previous knowledge of the application. The

Top-Down model tells us that Alya in MareNostrum 4 is affected by different factors when increasing `VECTOR_SIZE`.

Table 2. Top-Down model for Alya in MareNostrum 4

0.0 0.5 1.0

	1	2	4	8	16	32	64	128	256	512
Cycles [x10^{12}]	1.62	0.98	0.56	0.45	0.28	0.27	0.29	0.34	0.47	0.67
Instructions [x10^{12}]	3.64	2.63	1.31	0.96	0.57	0.49	0.45	0.45	0.44	0.45
IPC	2.24	2.67	2.33	2.15	2.01	1.77	1.57	1.32	0.94	0.67
Total Slots [x10^{12}]	6.47	3.93	2.25	1.80	1.13	1.10	1.15	1.38	1.92	2.65
├ Frontend Bound	0.16	0.13	0.16	0.14	0.07	0.06	0.04	0.04	0.02	0.02
│ ├ Fetch Latency	0.06	0.04	0.06	0.05	0.02	0.02	0.01	0.02	0.01	0.01
│ └ Fetch Bandwidth	0.10	0.09	0.10	0.08	0.05	0.04	0.03	0.02	0.01	0.01
├ Bad Speculation	0.03	0.04	0.06	0.07	0.11	0.10	0.10	0.09	0.06	0.04
│ ├ Branch Mispredicts	0.03	0.04	0.06	0.07	0.10	0.10	0.10	0.09	0.06	0.04
│ └ Machine Clears	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
├ Backend Bound	0.13	0.15	0.16	0.21	0.31	0.36	0.44	0.52	0.67	0.76
│ ├ Memory Bound	0.02	0.03	0.03	0.06	0.11	0.13	0.18	0.23	0.36	0.42
│ └ Core Bound	0.11	0.12	0.13	0.15	0.20	0.23	0.27	0.29	0.32	0.34
└ Retiring	0.68	0.68	0.61	0.59	0.51	0.47	0.42	0.36	0.25	0.18

1 to 32 the majority of the cycles are categorized as *Retiring*, which means that the pipeline is not stalled. The reader should note the model does not provide insight on whether the instructions that go through the pipeline are useful calculations or not.

32-128 the code is *Core Bound*, which means that the pipeline is stalled due to the computational resources. The official definition of TMAM defines metrics under *Core Bound*, but the PAPI installation in MareNostrum 4 does not provide access to the necessary counters. An instruction mix or pipeline usage breakdown could give more information about which type of operation is clogging the CPU. Note however, that the *Memory Bound* metric is gradually increasing. This means that the pipeline stalls due to memory accesses are becoming more impactful.

128-512 the code falls under *Memory Bound*, which means that slots are lost because the pipeline is waiting for memory resources. Again, the original model describes detailed metrics that we cannot measure in MareNostrum 4. Furthermore, we do not know which actions could the programmer take in order to reduce memory pressure.

Since we were not able to construct further levels of the Top-Down model in MareNostrum 4, we cannot investigate further the *Memory Bound* category. However, we know from our previous study that the issue with high values of `VECTOR_SIZE` is related to the memory, so we can study the number of accesses and misses to the different levels of the memory hierarchy. In the Skylake CPU, the L2 cache is the last level of private cache. It is the highest level in the memory hierarchy where we can measure memory accesses per core. Figure 9 shows the evolution of L2 cache accesses (green bars, measured with `PAPI_L2_TCA`), L2 cache misses (purple bars, measured with `PAPI_L2_TCM`) and L2 miss ratio (line, measured as `PAPI_L2_TCM/PAPI_L2_TCA`).

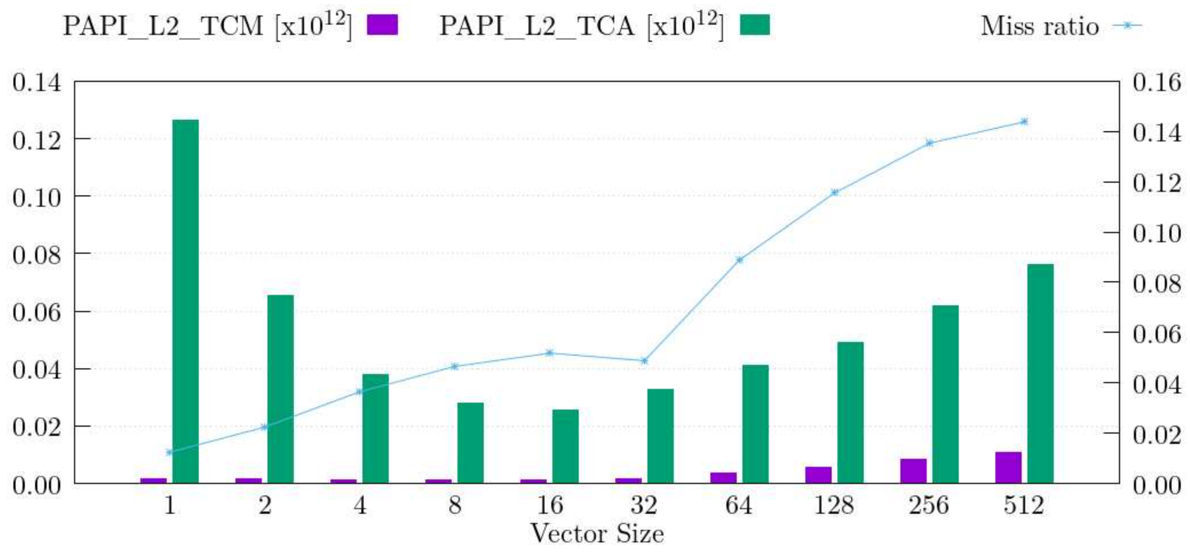

Figure 9. L2 accesses and misses in MareNostrum 4

We observe that the number of L2 cache accesses follows a *U* shape similar to the execution cycles. In contrast, the number of L2 cache misses stays flat from `VECTOR_SIZE` 1 to 32, and increases drastically for higher values. The combined view of L2 cache accesses and misses (miss ratio) shows that there is a noticeable jump starting at `VECTOR_SIZE` 32. This jump coincides with the point at which the elapsed time of Alya stops decreasing and the code modification appears to be detrimental. It also matches with the results presented in Table 2: the highest metric when `VECTOR_SIZE` is between 32 and 128 is *Core Bound*, but *Memory Bound* is the metric that is consistently increasing. We conclude that the code modification of Alya is beneficial in MareNostrum 4 up to `VECTOR_SIZE` 32, because it decreases the number of instructions executed and the number of L2 accesses. However, the modification reaches an inflexion point from which the L2 miss ratio becomes too high and the elapsed time bounces back up.

The Top-Down model has helped us to identify, in general terms, the part of the CPU that stalls during execution. However, more detailed metrics require access to hardware counters that are not accessible in our platform. This limits the insight that the model can provide. We can complement the information the Top-Down model gives us to compensate for the missing metrics (e.g., L2 accesses and misses). Without this complementary data, we cannot tell *how* execution cycles are lost. Furthermore, the study in MareNostrum 4 shows that only looking at the metric with the highest value does not tell the whole story.

5.2. CTE-AMD

Table 3 shows the Top-Down model for Alya in CTE-AMD. We observe a similar behavior as with MareNostrum 4 in Table 2 (i.e., the number of execution cycles decreases until `VECTOR_SIZE` 32, and bounces back up). In this case, we also identify the `Backend Bound` category to be the main limiting factor for high values of `VECTOR_SIZE`. However, the starting point of *Backend Bound* is 56% for CTE-AMD, while it is 13% for MareNostrum 4. Moreover, the portion of slots categorized as *Retiring* is always lower in CTE-AMD compared to MareNostrum 4. Unfortunately, we cannot drill further down because we have no definition of metrics nor more hardware counters available in the cluster.

Table 3. Top-Down model for Alya in CTE-AMD

0.0 0.5 1.0

	1	2	4	8	16	32	64	128	256	512
Cycles [x10^{12}]	1.32	0.93	0.62	0.54	0.43	0.45	0.43	0.44	0.52	1.50
Instructions [x10^{12}]	3.59	2.99	1.63	1.34	0.91	0.86	0.84	0.82	0.82	0.88
IPC	2.71	3.20	2.62	2.48	2.11	1.93	1.97	1.84	1.58	0.58
Total Slots [x10^{12}]	7.94	5.60	3.74	3.25	2.60	2.67	2.57	2.67	3.12	9.01
├ Frontend Bound	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.00
├ Bad Speculation	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.00
├ Backend Bound	0.56	0.47	0.57	0.59	0.64	0.67	0.66	0.68	0.73	0.90
└ Retiring	0.42	0.51	0.41	0.39	0.35	0.32	0.32	0.30	0.26	0.09

5.3. CTE-Arm

Table 3 shows the Top-Down model for Alya in CTE-Arm. The top part represents the first three levels of the model while the bottom represents the metrics under *Memory Bound*. Like with MareNostrum 4 and CTE-AMD, the execution cycles decrease up to a certain value of `VECTOR_SIZE` and then they jump back up. We observe that in CTE-Arm, the values of `VECTOR_SIZE` 128 and 256 yield the lowest amount of cycles. This is a much higher `VECTOR_SIZE` compared to MareNostrum 4 and CTE-AMD, which were around 32 and 64.

Table 4. Top-Down modelfor Alya in CTE-Arm

0.0 0.5 1.0

	1	2	4	8	16	32	64	128	256	512
Instructions [x10^{12}]	3.06	2.84	2.59	1.31	1.15	0.95	0.85	0.75	0.75	0.77
IPC	0.68	1.17	1.20	1.02	1.01	0.92	0.86	0.80	0.71	0.56
Total Cycles [x10^{12}]	4.47	2.42	2.16	1.28	1.13	1.03	0.98	0.93	1.06	1.38
├ 4INST_COMMIT	0.08	0.19	0.20	0.15	0.14	0.12	0.12	0.12	0.10	0.08
├ 3INST_COMMIT	0.05	0.06	0.06	0.07	0.08	0.08	0.06	0.05	0.04	0.03
├ 2INST_COMMIT	0.05	0.06	0.05	0.06	0.06	0.06	0.05	0.05	0.05	0.04
├ 1INST_COMMIT	0.08	0.08	0.09	0.09	0.09	0.09	0.08	0.08	0.07	0.06
└ 0INST_COMMIT	0.73	0.60	0.59	0.63	0.63	0.66	0.68	0.70	0.72	0.79
├ Frontend Bound	0.09	0.06	0.05	0.05	0.04	0.03	0.03	0.03	0.03	0.02
├ Bad Speculation	0.02	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.02
├ Memory Bound	0.42	0.46	0.48	0.50	0.49	0.56	0.61	0.67	0.69	0.75
├ Compute Bound	0.43	0.40	0.37	0.35	0.38	0.33	0.30	0.27	0.24	0.20
├ Complex Instr.	0.04	0.05	0.06	0.07	0.06	0.05	0.03	0.01	0.01	0.01
└ MOVPRX Instr.	0.01	0.01	0.00	0.01	0.00	0.00	0.00	-0.01	0.01	0.00

	1	2	4	8	16	32	64	128	256	512
Memory Bound	0.42	0.46	0.48	0.50	0.49	0.56	0.61	0.67	0.69	0.75
├ Base instructions	0.35	0.52	0.48	0.51	0.48	0.49	0.45	0.42	0.43	0.49
├ L1 misses	0.13	0.20	0.15	0.21	0.19	0.25	0.30	0.35	0.33	0.44
├ L1 hit	0.48	0.21	0.30	0.20	0.26	0.09	0.09	0.09	0.11	-0.01
├ L2 prefetch busy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
└ L2 misses	0.04	0.08	0.07	0.08	0.08	0.17	0.16	0.15	0.12	0.08

The Top-Down model in CTE-Arm does not have a *Retiring* metric, like in MareNostrum 4. We can achieve a similar metric by combining *{4,3,2,1}INST_COMMIT*. This new metric groups cycles where at least one instruction was committed, which is similar to the definition of *Retiring* but includes cycles where the CPU could not commit some instruction. The key difference between the model in MareNostrum 4 in CTE-Arm that makes it impossible to have a comparable *Retiring* is that the first uses *Slots* to construct its metrics while the second uses *Cycles*.

Looking at the other end of the spectrum, we observe that for `VECTOR_SIZE` 1, CTE-Arm spends 75% of the execution cycles completely stalled (*0INST_COMMIT*). This trend is observable across all values of `VECTOR_SIZE`, with 4 showing the lowest (59%) value and 512 showing the highest (79%). The reader should note that the *0INST_COMMIT* metric is indicative of how bad the pipeline is stalled, but it does not reflect the total execution time. Furthermore, the metric is relative to the total execution cycles, which means that the run with the lowest *0INST_COMMIT* is not necessarily the fastest. As it stands, the Top-Down model in CTE-Arm does not allow us to compare metric-to-metric different runs. What it allows us to do, is to walk down the hierarchy in a particular run or observe general trends across runs.

For low values of `VECTOR_SIZE`, the metrics *Memory Bound* and *Compute Bound* are the main limiting factors. From `VECTOR_SIZE` 32 onward, *Memory Bound* accounts for more than half of the cycles lost in *0INST_COMMIT*. We can further drill down and measure the metrics bellow *Memory Bound* (shown in the bottom part of Table 3). We observe that *Base instructions* is always the dominant metric, with the exception of `VECTOR_SIZE` 1 where *L1 hit* is higher. The description of this metric in the official documentation by Fujitsu [11] states: *Cycles caused by instructions belonging to Base Instructions*[2]. With only this definition, it is unclear if there is overlap between the metric and other metrics under *Memory Bound*, but Table 14-6 of the same document verifies that there is no overlap. Our observation is that there is an increase of the proportion of cycles lost due to *L1 misses* while the inverse happens for *L1 hit*. With the information available, we cannot blame the cycles lost waiting for the completion of a memory access *Memory Bound* to a specific type of load operation, we can only conclude that it is a scalar instruction (not SIMD nor SVE).

5.4. CTE-Power

Table 5 shows the Top-Down model for Alya in CTE-Power. Similar to CTE-Arm, the number of total cycles decreases while `VECTOR_SIZE` increases, but they start bouncing back up when `VECTOR_SIZE` is 512.

Throughout all the executions, the main limiting factor is the *Pipeline stall* category, which is comparable to the *Retiring* category in MareNostrum 4. Furthermore, the *Memory* subcategory represents always between 85% and 90% of the stalled cycles. In contrast, the *Compute* subcategory, the second highest, only accounts 15% max.

[2] The set of *Base Instructions* in the Armv8 ISA contains basic scalar arithmetic and memory instructions.

Table 5. Top-Down model for Alya in CTE-Power

0.0 0.5 1.0

Total Cycles [x10^{12}]	7.08	4.10	2.54	1.78	1.38	1.20	1.07	1.03	1.01	1.08
├ ICT empty	0.03	0.03	0.03	0.03	0.03	0.03	0.02	0.02	0.02	0.03
├ Issue hold	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
├ Pipeline stall	0.60	0.59	0.56	0.55	0.53	0.53	0.53	0.53	0.53	0.56
│ ├ Branches	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
│ ├ Compute	0.11	0.12	0.14	0.15	0.15	0.13	0.13	0.13	0.13	0.11
│ ├ Memory	0.89	0.88	0.86	0.85	0.85	0.87	0.87	0.87	0.87	0.89
│ └ Other	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
├ Thread blocked	0.10	0.10	0.09	0.08	0.07	0.05	0.05	0.04	0.04	0.04
├ Instruction latency	0.14	0.15	0.15	0.17	0.19	0.21	0.22	0.22	0.23	0.21
└ Completion cycles	0.13	0.14	0.16	0.17	0.18	0.18	0.18	0.18	0.18	0.16
	1	2	4	8	16	32	64	128	256	512

Memory	0.89	0.88	0.86	0.85	0.85	0.87	0.87	0.87	0.87	0.89
├ LS addr. queue	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02
├ ERAT miss queue	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.01
├ DCache miss	0.07	0.09	0.11	0.12	0.11	0.10	0.11	0.12	0.12	0.21
├ Load finish	0.19	0.19	0.19	0.21	0.18	0.15	0.15	0.15	0.14	0.13
├ Store reorder queue	0.50	0.46	0.42	0.34	0.28	0.21	0.19	0.16	0.16	0.14
├ Store finish	0.16	0.19	0.21	0.27	0.37	0.49	0.52	0.54	0.55	0.48
├ LS unit	0.06	0.05	0.05	0.04	0.03	0.02	0.02	0.02	0.02	0.01
└ Other	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	1	2	4	8	16	32	64	128	256	512

Drilling into the *Memory* category, we observe that there are two metrics which evolve noticeably when increasing `VECTOR_SIZE`: *Store reorder queue* and *Store finish*. While the first one represents 50% of the cycles due to memory stalls with `VECTOR_SIZE` 1, its weight decreases while the weight of *Store finish* increases.

- *Store reorder queue* is defined by the counter `PM_CMPLU_STALL_SRQ`, which measures the cycles a store operation was stalled because the store-reorder-buffer (SRQ) was full (i.e., Too many store operations were in-flight at the same time).
- *Store finish* is defined by the counter `PM_CMPLU_STALL_STORE_FINISH`, which measures the cycles waiting for a store operation that has all of its dependencies met to finish (i.e., The nominal latency of a store operation).

Following the definition of both metrics, we can infer that the parameter `VECTOR_SIZE` has an effect on the amount of store operations that are in-flight at the same time. For low values of the parameter (`VECTOR_SIZE` ≤ 16), there are too many stores at once, which stalls the pipeline. From that point forward, the concurrent store operations decrease, which means that the pipeline is still waiting for stores to complete, but these have a minimum amount of cycles to complete (pipeline latency). The Top-Down model in CTE-Power is heavily dependent on the micro-architecture, so going deep into the hierarchy might give more insight about performance bottlenecks, but makes it harder to compare against other machines.

5.5. *CTE-Kunpeng*

Table 6 shows the Top-Down model for Alya in CTE-Kunpeng. Furthermore, the model has been defined to be very similar to the original Top-Down model from Intel. Contrary to MareNostrum 4, the results in CTE-Kunpeng show that *Core Bound* is the main limiting factor for low values of `VECTOR_SIZE`. We know that the CPU in CTE-Kunpeng has a lower single-thread performance compared to MareNostrum 4 (see Table 1). In the case of Alya, the weaker floating-point throughput of the core is reflected as cycles stalled due to the computational resources being occupied. At the time of writing, we do not have the definition of the metrics under *Core Bound*, so we cannot construct the whole hierarchy.

Table 6. Top-Down model for Alya in CTE-Kunpeng

	1	2	4	8	16	32	64	128	256	512
Cycles [x10^{12}]	3.00	1.96	1.30	1.08	0.96	0.90	1.02	1.34	1.66	1.97
Instructions [x10^{12}]	7.05	4.06	2.92	2.37	2.03	1.80	2.11	2.10	2.11	2.14
IPC	2.35	2.07	2.25	2.19	2.10	2.00	2.07	1.57	1.27	1.09
Total Slots [x10^{12}]	12.00	7.84	5.20	4.32	3.86	3.59	4.08	5.35	6.66	7.88
├ Frontend Bound	0.20	0.23	0.16	0.14	0.14	0.13	0.11	0.08	0.06	0.06
├ Bad Speculation	0.06	0.12	0.03	0.02	0.02	0.03	0.03	0.02	0.01	0.02
├ Backend Bound	0.15	0.14	0.24	0.29	0.32	0.34	0.34	0.51	0.61	0.65
│ ├ Memory Bound	0.15	0.16	0.13	0.11	0.12	0.17	0.25	0.44	0.53	0.59
│ │ ├ L1 Bound	0.13	0.13	0.09	0.07	0.06	0.06	0.06	0.05	0.06	0.05
│ │ ├ L2 Bound	0.14	0.14	0.09	0.08	0.08	0.08	0.08	0.07	0.08	0.09
│ │ └ L3/DRAM	0.02	0.02	0.04	0.03	0.04	0.09	0.16	0.33	0.41	0.47
│ └ Core Bound	0.85	0.84	0.87	0.89	0.88	0.83	0.75	0.56	0.47	0.41
└ Retiring	0.59	0.52	0.56	0.55	0.53	0.50	0.52	0.39	0.32	0.27

As `VECTOR_SIZE` increases, so does the *Memory Bound* metric. Starting at `VECTOR_SIZE` 256, Alya has become bounded by the memory subsystem. For the *Memory Bound* part, we can study the different cache levels in CTE-Kunpeng. This in-depth study was not possible in MareNostrum 4 since we lacked the PAPI counters to compute the metrics. At this point, the model suggest that cycles are being lost due to L3 Cache and DRAM accesses or misses. The cache hierarchy of the Kunpeng 920 CPU shares the L3 level across cores, which makes it difficult to blame accesses or misses to a particular core.

6. Conclusions

In this paper, we have studied the Top-Down model for three different ISAs (x86-64, Arm-v8 and IBM Power9) implemented by five different chip providers (Intel, AMD, Fujitsu, Huawei and IBM) in five HPC clusters. We used Alya, a CFD application, as a vehicle to measure the metrics of each model and gain insights about performance bottlenecks. The results of our study can be summarized into two main categories: *i)* conclusions that can be drawn when studying the top-down model within the same cluster and *ii)* considerations that relate to the top-down results gathered from different clusters.

Within the same cluster we found that the implementation of the Top-Down model can be tricky: going deep into the hierarchy of the model means needing hardware counters, which are not always available due to limitation in tools maturity or system software configuration. Also, some architectural details can be missing.

On a case-by-case, x86-64 Skylake is the most documented and with official support by Intel. The metrics are well defined and easy to understand, but they require hardware counters that are not available in MareNostrum 4. This issue is even more apparent in CTE-AMD, where we were not able to go past the first level of metrics of the Top-Down model. In the case of CTE-Power, the amount of counters needed to construct the whole hierarchy implies obtaining the data with multiple runs. Once constructed, interpreting the metrics requires a deeper understanding of the micro-architecture of the system compared to the other clusters. The model in CTE-Arm tries to strike a balance: the official documentation accompanies the *Cycle Accounting* with multiple tables of complementary metrics so we were able to go beyond the first level of metrics. However, the model definition in CTE-Arm does not allow for metric comparisons between runs because metrics are relative to their parent in the tree and not to the root.

Once mapped the hardware counters on the Top-Down model in a given system, we comment about the insights we can obtain from the model itself. The metrics are always defined relative to the execution cycles (or parent metric) of their respective runs. Thus, when comparing runs performed with different software configurations (e.g., changing `VECTOR_SIZE`) on the same cluster, having a higher or lower value of a certain metric does not imply a better or worse execution time. The Top-Down model only shows general trends and how this evolves throughout different runs with different configurations (e.g., changing `VECTOR_SIZE`). Once constructed, the interpretation of the metrics in all clusters depends on micro-architectural knowledge.

Among different clusters we discover even more limitations. Having no common naming-convention of metrics across systems, makes it hard to compare the results of Top-Down models gathered on different clusters. Even with the same names, metrics in different systems have subtle differences: e.g., on MareNostrum 4, CTE-AMD, and CTE-Kunpeng the inefficiencies of the Top-Down model are gathered on the boundary between front-end and back-end of the CPU, while on CTE-Arm and CTE-Power the model counts the cycles lost during the commit stage of the instructions.

We have seen that on different machines we can go deeper in the hierarchy of the model, depending on availability of hardware counters. However, even having access to the whole Performance Monitoring Unit (where hardware counters are stored) does not magically mean having more insight because a correct interpretation of the most of the counters requires a deep knowledge of the underlying micro-architecture that the scientist running a scientific code does not have. This creates a trade-off when using the Top-Down model: The deeper the detail and insight, the more it depends on previous micro-architecture knowledge and the more difficult it becomes to compare against other machines. Moreover, in some clusters the definition of the model requires architectural parameters which makes more difficult the work of defining the model and comparing with other clusters.

As general and most important remark, we conclude that there is no metric in none of the models that tell *how much* the CPU resources are being used. One could measure 100% of *Slots* in the *Retiring* category, but that does not tell if the code is running efficiently: it simply tells that there are no pipeline stalls. In the Top-Down model defined by Intel, there are some metrics under the *Retiring* category which try to tell if there is room for improvement. However, we did not include these metrics in our work because (once more) we were limited by the available hardware counters.

While the Top-Down model is accepted by the community as a method for spotting inefficiencies of HPC codes, we have experienced that:

1. it requires additional information (either micro-architectural details or further hardware counters) to draw a full picture when analyzing and improving the performance of a scientific application;
2. it does not quantify how much each of the resources within the compute node are used/saturated;
3. it does not allow to easily compare among clusters of the same architecture, nor different architectures.

All these limitations makes the work very difficult to a performance analyst that wants to use the Top-Down model as an inspection tool for spotting inefficiencies in complex scientific codes. The dependencies with the micro-architectural knowledge makes it also not feasible to be proposed as co-design tool to the researcher owner of the scientific application under study.

Author Contributions: All authors have contributed equally to the work reported.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: The authors thank the support team of Dibona operating at ATOS/Bull. This work is partially supported by the Spanish Government through Programa Severo Ochoa (SEV-2015-0493), the Spanish Ministry of Science and Technology project (TIN2015-65316-P), the Generalitat de Catalunya (2017-SGR-1414), the European Community's Seventh Framework Programme [FP7/2007-2013] and Horizon 2020 under the Mont-Blanc projects (grant agreements n. 288777, 610402 and 671697), the European POP2 Center of Excellence (grant agreement n. 824080), and the Human Brain Project SGA2 (grant agreement n. 785907).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Event sets and metrics

In Table A1, we list the PAPI Event sets used to construct the Top-Down model on each system under study. The left column displays the name of the event set, while the right column lists the PAPI events included in a given set. Events repeated in more that one set are omitted.

Table A1. Event set reference for MareNostrum 4, CTE-AMD, CTE-Arm, CTE-Power, and CTE-Kunpeng

MareNostrum 4	
EventSet1	UOPS_RETIRED:RETIRE_SLOTS, BR_MISP_RETIRED:ALL_BRANCHES MACHINE_CLEARS:COUNT, EXE_ACTIVITY:BOUND_ON_STORES
EventSet2	CPU_CLK_THREAD_UNHALTED, CYCLE_ACTIVITY:STALLS_MEM_ANY IDQ_UOPS_NOT_DELIVERED:CORE IDQ_UOPS_NOT_DELIVERED:CYCLES_0_UOPS_DELIV_CORE UOPS_ISSUED:ANY, INT_MISC:RECOVERY_CYCLES
CTE-AMD	
EventSet1	UOPS_QUEUE_EMPTY, RETIRED_BRANCH_INSTRUCTIONS_MISPREDICTED RETIRED_TAKEN_BRANCH_INSTRUCTIONS_MISPREDICTED RETIRED_INDIRECT_BRANCH_INSTRUCTIONS_MISPREDICTED
EventSet2	CYCLES_NOT_IN_HALT, RETIRED_UOPS, RETIRED_INSTRUCTIONS
CTE-Arm	
EventSet1	CPU_CYCLES, 0INST_COMMIT, 1INST_COMMIT 2INST_COMMIT, 3INST_COMMIT, 4INST_COMMIT
EventSet2	LD_COMP_WAIT, EU_COMP_WAIT, BR_COMP_WAIT ROB_EMPTY, UOP_ONLY_COMMIT, SINGLE_MOVPRFX_COMMIT
EventSet3	LD_COMP_WAIT_EX, LD_COMP_WAIT_L2_MISS LD_COMP_WAIT_L2_MISS_EX, LD_COMP_WAIT_L1_MISS LD_COMP_WAIT_L1_MISS_EX, LD_COMP_WAIT_PFP_BUSY
CTE-Power	
EventSet1	PM_RUN_CYC, PM_CMPLU_STALL_BRU, PM_NTC_ISSUE_HELD_ARB PM_NTC_ISSUE_HELD_DARQ_FULL, PM_NTC_ISSUE_HELD_OTHER
EventSet2	PM_CMPLU_STALL_EXEC_UNIT, PM_CMPLU_STALL_NTC_DISP_FIN PM_CMPLU_STALL_SRQ_FULL, PM_ICT_NOSLOT_CYC
EventSet3	PM_CMPLU_STALL_EMQ_FULL, PM_CMPLU_STALL_LOAD_FINISH PM_CMPLU_STALL_NTC_FLUSH, PM_CMPLU_STALL_THRD
EventSet4	PM_1PLUS_PPC_CMPL, PM_CMPLU_STALL_DCACHE_MISS PM_CMPLU_STALL_LMQ_FULL, PM_CMPLU_STALL_LSU_MFSPR
EventSet5	PM_CMPLU_STALL_LARX, PM_CMPLU_STALL_LRQ_FULL PM_CMPLU_STALL_LSAQ_ARB, PM_CMPLU_STALL_STORE_DATA
EventSet6	PM_CMPLU_STALL_ERAT_MISS, PM_CMPLU_STALL_HWSYNC PM_CMPLU_STALL_LHS, PM_CMPLU_STALL_LRQ_OTHER
EventSet7	PM_CMPLU_STALL_LSU_FIN, PM_CMPLU_STALL_ST_FWD PM_CMPLU_STALL_STORE_FIN_ARB, PM_CMPLU_STALL_STORE_FINISH
EventSet8	PM_CMPLU_STALL_EIEIO, PM_CMPLU_STALL_SLB, PM_CMPLU_STALL_TLBIE
EventSet9	PM_CMPLU_STALL_LWSYNC, PM_CMPLU_STALL_PASTE PM_CMPLU_STALL_STORE_PIPE_ARB
EventSet10	PM_CMPLU_STALL_STCX, PM_CMPLU_STALL_TEND
CTE-Kunpeng	
EventSet1	INST_RETIRED, CPU_CYCLES FETCH_BUBBLE, INST_SPEC
EventSet2	MEM_STALL_ANYLOAD, MEM_STALL_ANYSTORE, EXE_STALL_CYCLE MEM_STALL_L1MISS, MEM_STALL_L2MISS

In Table A2, we list the PAPI counters and formulas used to construct the Top-Down model on each system under study. The left column displays the name of a metric, while the right column details the formula to compute the metric. Counters are written in `monospace` font and using the exact same name as displayed by the `papi_native_avail` command. Metrics of the model as well as helper metrics are written in *cursive*.

Table A2. Top-Down model counter reference for MareNostrum 4, CTE-AMD, CTE-Arm, CTE-Power, and CTE-Kunpeng

MareNostrum 4	
Pipeline_Width	4
Clocks	`CPU_CLK_THREAD_UNHALTED`
Slots	*Pipeline_Width* × *Clocks*
Frontend_Bound	`IDQ_UOPS_NOT_DELIVERED:CORE` / *Slots*
Fetch_Latency	*Pipeline_Width* × `IDQ_UOPS_NOT_DELIVERED:CYCLES_0_UOPS_DELIV_CORE` / *Slots*
Fetch_Bandwidth	*Frontend_Bound* - *Fetch_Latency*
Bad_Speculation	(`UOPS_ISSUED:ANY` - `UOPS_RETIRED:RETIRE_SLOTS` + *Pipeline_Width* × `INT_MISC:RECOVERY_CYCLES`) / *Slots*
Branch_Mispredicts	*Mispred_Clears_Fraction* × *Bad_Speculation*
Machine_Clears	*Bad_Speculation* - *Branch_Mispredicts*
Mispred_Clears_Fraction	`BR_MISP_RETIRED:ALL_BRANCHES` / (`BR_MISP_RETIRED:ALL_BRANCHES` + `MACHINE_CLEARS:COUNT`)
Backend_Bound	1 - *Frontend_Bound* - (`UOPS_ISSUED:ANY` + *Pipeline_Width* × `INT_MISC:RECOVERY_CYCLES`)/ *Slots*
Memory_Bound	*Memory_Bound_Fraction* × *Backend_Bound*
Core_Bound	*Backend_Bound* - *Memory_Bound*
Memory_Bound_Fraction	`CYCLE_ACTIVITY:STALLS_MEM_ANY` + `EXE_ACTIVITY:BOUND_ON_STORES`) / `Backend_Bound_Cycles`
Retiring	`UOPS_RETIRED:RETIRE_SLOTS` / *Slots*
CTE-AMD	
Pipeline_Width	6
Mispredict_Cost	18
Clocks	`CYCLES_NOT_IN_HALT`
Slots	*Pipeline_Width* × *Clocks*
Frontend_Bound	`UOPS_QUEUE_EMPTY` / *Slots*
Bad_Speculation	*Branch_Instructions* × *Mispred_Cost* / *Slots*
Branch_Instructions	`RETIRED_BRANCH_INSTRUCTIONS_MISPREDICTED` + `RETIRED_INDIRECT_BRANCH_INSTRUCTIONS_MISPREDICTED` + `RETIRED_TAKEN_BRANCH_INSTRUCTIONS_MISPREDICTED`
Backend_Bound	1 - (*Frontend_Bound* + *Bad_Speculation* + *Retiring*)
Retiring	`RETIRED_UOPS` / *Slots*
CTE-Arm	
Clocks	`CYCLES_NOT_IN_HALT`
4_Instruction_Commit	`4INST_COMMIT` / *Clocks*
3_Instruction_Commit	`3INST_COMMIT` / *Clocks*
2_Instruction_Commit	`2INST_COMMIT` / *Clocks*
1_Instruction_Commit	`1INST_COMMIT` / *Clocks*
0_Instruction_Commit	`0INST_COMMIT` / *Clocks*
Frontend_Bound	`ROB_EMPTY` / `0_INST_COMMIT`
Bad_Speculation	`BR_COMP_WAIT` / `0_INST_COMMIT`
Memory_Bound	`LD_COMP_WAIT` / `0_INST_COMMIT`
Compute_Bound	`EU_COMP_WAIT` / `0_INST_COMMIT`
Complex_Instructions	`UOP_ONLY_COMMIT` / `0_INST_COMMIT`
MOVPRX_Instructions	`SINGLE_MOVPRX_COMMIT` / `0_INST_COMMIT`
CTE-Power	
Clocks	`PM_RUN_CYC`
No_Instruction_To_Execute	`PM_ICT_NOSLOT_CYC` / *Clocks*
Instruction_Held_In_Issue	`PM_ISSUE_HOLD` / *Clocks*
Backend_Bound	`PM_CMPLU_STALL` / *Clocks*
Stalled_By_Other_Thread	`PM_CMPLU_STALL_THRD` / *Clocks*
???	`PM_1PLUS_PPC_CMPL` / *Clocks*
Completion_Cycles	1 - (*No_Instruction_To_Execute* + *Instruction_Held_In_Issue* + *Backend_Bound* + *Stalled_By_Other_Thread* + *???*)
CTE-Kunpeng	
Pipeline_Width	4
Clocks	`CPU_CYCLES`
Slots	*Pipeline_Width* × *Clocks*
Frontend_Bound	`FETCH_BUBBLE` / *Slots*
Bad_Speculation	(`INST_SPEC` - `INST_RETIRED`) / *Slots*
Backend_Bound	1 - (*Frontend_Bound* + *Bad_Speculation* + *Retiring*)
Memory_Bound	*Memory_Stall_Cycles* / `EXE_STALL_CYCLE`
Core_Bound	(`EXE_STALL_CYCLE` - *Memory_Stall_Cycles*) / `EXE_STALL_CYCLE`
Memory_Stall_Cycles	`MEM_STALL_ANYLOAD` + `MEM_STALL_ANYSTORE`
Retiring	`INST_RETIRED` / *Slots*

References

1. Top500 list, 2022.

2. Williams, S.; Waterman, A.; Patterson, D. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* **2009**, *52*, 65–76. doi:10.1145/1498765.1498785.
3. Ofenbeck, G.; Steinmann, R.; Caparros, V.; Spampinato, D.G.; Püschel, M. Applying the roofline model. 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2014, pp. 76–85.
4. Ilic, A.; Pratas, F.; Sousa, L. Cache-aware Roofline model: Upgrading the loft. *IEEE Computer Architecture Letters* **2014**, *13*, 21–24. doi:10.1109/L-CA.2013.6.
5. Banchelli, F.; Garcia-Gasulla, M.; Houzeaux, G.; Mantovani, F. Benchmarking of State-of-the-Art HPC Clusters with a Production CFD Code. Proceedings of the Platform for Advanced Scientific Computing Conference; Association for Computing Machinery: New York, NY, USA, 2020; PASC '20. doi:10.1145/3394277.3401847.
6. Yasin, A. A Top-Down method for performance analysis and counters architecture. 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014, pp. 35–44. doi:10.1109/ISPASS.2014.6844459.
7. Intel. Top-down Microarchitecture Analysis Method, 2022.
8. Jarus, M.; Oleksiak, A. Top-Down Characterization Approximation based on performance counters architecture for AMD processors. *Simulation Modelling Practice and Theory* **2016**, *68*, 146–162. doi:10.1016/j.simpat.2016.08.006.
9. Banchelli, F.; Oyarzun, G.; Garcia-Gasulla, M.; Mantovani, F.; Both, A.; Houzeaux, G.; Mira, D. A portable coding strategy to exploit vectorization on combustion simulations, 2022, [arXiv:cs.DC/2210.11917].
10. Fog., A. The microarchitecture of Intel, AMD, and VIA CPUs - An optimization guide for assembly programmers and compiler makers, 2022.
11. A64FX Microarchitecture Manual, 2021.
12. POWER9 Performance Monitor Unit User's Guide, 2018.
13. Unified European Applications Benchmark Suite. http://www.prace-ri.eu/ueabs/ - Last accessed Nov. 2016.
14. Terpstra, D.; Jagode, H.; You, H.; Dongarra, J. Collecting Performance Data with PAPI-C. Tools for High Performance Computing 2009; Müller, M.S.; Resch, M.M.; Schulz, A.; Nagel, W.E., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2010; pp. 157–173.
15. Pillet, V.; Pillet, V.; Labarta, J.; Cortes, T.; Cortes, T.; Girona, S.; Girona, S.; Computadors, D.D.D. PARAVER: A Tool to Visualize and Analyze Parallel Code. Technical report, In WoTUG-18, 1995.