Folding: reporting instantaneous performance metrics and source-code references

Harald Servat, Jesús Labarta
BSC – Computer Sciences Department
UPC-DAC – Computer Architecture Department

***Abstract*-** *Despite supercomputers deliver huge computational power, applications only reach a fraction of it. There are several factors limiting the application performance, and one of the most important is the single processor efficiency because it ultimately dictates the overall achieved performance. We present the folding mechanism, a process that combines measurements captured through minimal instrumentation and coarse-grain sampling ensuring low time dilation (less than 5%). The mechanism reports instantaneous performance and source-code references for optimized binaries accurately by taking advantage of the repetitiveness of many applications, especially in HPC. The mechanism enables the exploration of the application performance and guides the analyst to source-code modifications.*

I. Motivation

Nowadays, supercomputers deliver an enormous amount of computational power; however, it is well-known that applications only reach a fraction of it. There are several factors that limit the achieved application performance. One of the most important factors is the single processor performance (*i.e.* how fast a processor executes a work unit) because it ultimately dictates the overall achieved performance. Performance analysis tools are pieces of software that help locating performance inefficiencies and identifying their nature within applications to ultimately improve the application performance. These tools rely on two collection techniques to invoke their performance monitors: instrumentation and sampling. Instrumentation refers to the ability to inject performance monitors into concrete application locations whereas sampling deals with invoking the installed monitors periodically. Each technique has its advantages. The measurements obtained through instrumentation are directly associated with the application structure while sampling allows a simple way to determine the volume of the measurements captured. In any case, the granularity of the measurements provides valuable insight cannot be easily determined *a priori*. Should analysts study the performance of an application for the first time, they may consider using a performance tool and instrument every routine or use high-frequency sampling rates in order to provide the most detailed results. More often than not, these approaches lead to large overheads that impact on the application performance and thus alter the measurements gathered and, therefore, mislead the analyst.

II. Folding

The folding mechanism overcomes the overhead by taking advantage of the repetitiveness of many applications. This mechanism smartly combines instrumented and inexpensive coarse-grain sampled information that dilates the application runtime less than 5% on optimized binaries. The results of the mechanisms include rich reports that show the instantaneous performance evolution and source-code progression within instrumented regions of code accurately. To this end, the captured metrics should contain information regarding performance metrics associated to the processor such as number of instructions executed, number of L1 cache misses, and stalled cycles, as well as, call-stack information to correlate with the application structure.

The folding projects the collected samples into a synthetic instance preserving their time since the start of their respective instance; thus, a sample fired at time Ts within an instance that starts at Ti gets mapped into the representative region at time δs, where δs = Ts - Ti. Figure 2 provides a visual description of the process in order to help understand it. The top of the Figure depicts a time-line of an application with a repetitive region (e.g. a routine, a loop body, …) that has been executed three times during the whole execution whereas the bottom part schematizes how the folding works. The analyst instruments the application to determine when each routine invocation begins (shown as Ix, where x={1,2,3}) and ends (shown as I'x, where x={1,2,3}). The analyst also enables a periodic sampling mechanism that freely runs no matter the application activity (in the example, providing measurements every seven units of time). As a consequence of using the folding mechanism, the synthetic instance contains more samples than any other instance, and therefore, it is capable of depicting the progression within the instance more accurately.

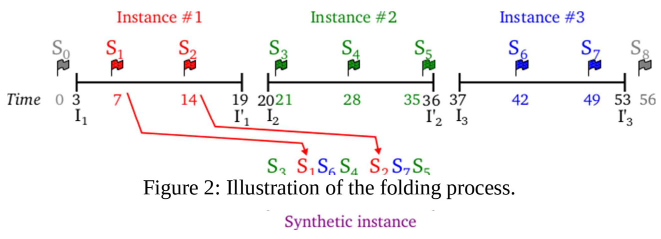
Instance #1
Instance #2
Instance #3
S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8
Time 0 3 7 14 19 20 21 28 35 36 37 42 49 53 56
I_1 I'_1 I_2 I'_2 I_3 I'_3
S_3 S_1 S_6 S_4 S_2 S_7 S_5
Synthetic instance

Figure 2: Illustration of the folding process.

A. Detailed performance counter evolution

The folding depicts instantaneous progression of the performance counters using a fitting mechanism on the performance metrics associated to the folded samples. We have evaluated two fitting algorithms used in different areas to report continuous metrics: a Gaussian interpolation process named Kriging [1], and piece-wise linear regressions. No matter the approach taken, the folding results include the progression of the performance counter rates. From all these counter rates, the MIPS rate indicates whether a region of code runs at good pace but it does not unveil the reasons for performance bottlenecks, if any. Our work has addressed this topic by taking advantage of performance analytical models based on performance counters and applying them to the folding results. For instance, Figure **3** shows the progression of the MIPS rate (in black, on the right Y-axis) and other performance counter ratios per instruction (in other colors, on the left Y-axis) in the most time-consuming region of CGPOP [2]. The plot exposes two phases within this region. The first phase runs approx. at 500 MIPS and seems to be limited by the amount of cache misses per instruction (approx. 6%). The second phase starts running at 2000 MIPS after decreasing the cache miss ratio from the first phase. The subsequent decrease of L3 cache misses turns into an increase to the instruction rate up to 3500 MIPS.

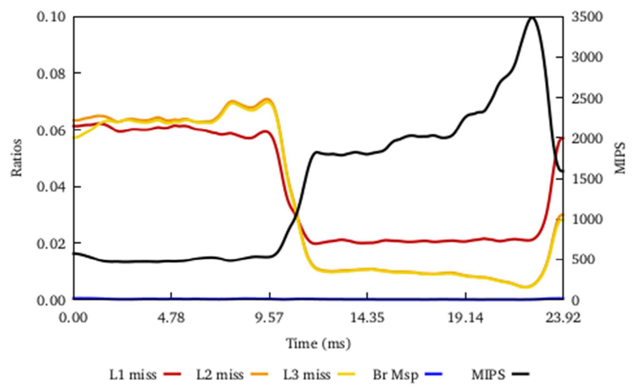

Figure 3: Folding results for the most time-consuming region in the CGPOP application.

B. Detailed correlation with source-code

The correlation of performance inefficiencies and their associated source code is a cornerstone to understand why the efficiency of an application falls behind the computer's peak performance and to enable optimizations on the application code, ultimately. We have explored the opportunities of the folding mechanism when treating source code references captured in the samples to allow the analyst easily understand the application's bottlenecks. Despite source code references do not benefit from fitting models as the performance counters, our work has also explored two approaches to establish an approximate correlation between performance and source code. One of these approaches takes benefit of the phases delimited by the piece-wise linear regressions when fitting the performance counters, and associates source code references to each phase. The other approach has been inspired by MultiSequence Alignment techniques [3,4] and finds similarities in consecutive sampled call-stacks to unveil the active routine.

Figure 4 exemplifies the results in which the performance results are collocated with their source-code when exploring one of the most time-consuming regions in Arts_CF [5]. This region exposes two phases in terms of the instruction rate and each of these phases is correlated with a particular region of code (shown in a different background color). It is worth to notice that the first phase (shown in red), reveals that the routine scalar_weno5_coeff (line 667) runs approx. at 2600 MIPS, which is far below the processor's peak performance. Despite we did not have access to the application source-code, we asked the developer for the surrounding lines and after exploring them we were able to suggest optimizations that improved the performance of the region by 17.4% by applying well-known optimization techniques to tiny regions of code.

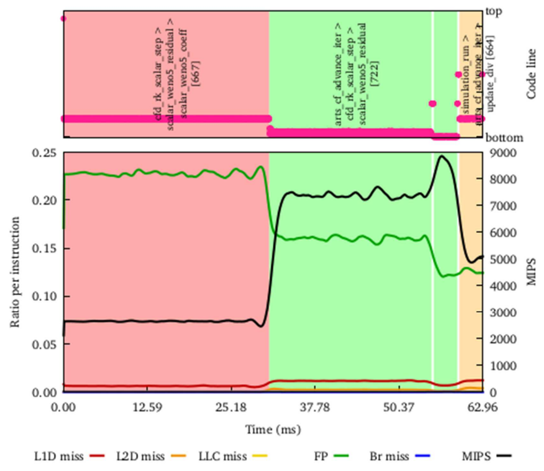

Figure 4: Results of the Arts_CF application showing the source-code collocated with the performance results.

Conclusions

We have presented a mechanism that reports the instantaneous progression of the performance metrics and source code references accurately for already optimized binaries. The mechanism is capable of reporting the performance progression of very small delimited regions of code by using coarse-grain sampling, ensuring low intrusion during the application execution. Despite not every application may benefit from manually changing the source-code, the mechanism has proven valuable to guide the analyst to small code changes based on well-known optimization techniques to further increase the performance.

References

[1] F. Trochu. "A contouring program based on dual kriging implementation." Engineering with computers 9.4 (1,993), pp. 160-177.

[2] Andrew Stone, John Dennis and Michelle Mills-Strout. "The CGPOP miniapp, v1.0" Tech report CS-11-103- Colorado State University 2,011.

[3] Cedric Notredame, Desmond G. Higgins and Jaap Hering. "T-Coffee: a novel method for fast and accurate Multiple-Sequence Alignment" Journal of molecular biology 302.1 (2,000) pp 205-217.

[4] Andreas Doring, David Weese, Tobias Rausch and Knut Reiner. "SeqAn an efficient, generic C++ library for sequence analysis" BMC bioinformatics 9.1 (2,009) pp 11.

[5] Olivier Desjardins, Guillaume Blanquart, Guillaume Balarac, and Heinz Pitsch. “High order conservative finite difference scheme for variable density low Mach number turbulent flows” Journal of Computational Physics 227.15 (2,008), pp-7125-7159.

LIST OF PUBLICATIONS

Harald Servat, Germán Llort, Judit Giménez, Jesús Labarta “Detailed Performance analysis using Coarse Grain Sampling” In proceeding of the Workshop on Productivity and Performance (PROPER), in conjunction with Euro-PAR 2,009: 185-198.

Harald Servat, Germán Llort, Judit Giménez, Kevin A. Huck, Jesús Labarta “Unveiling Internal Evolution of Parallel Application Computation Phases” In proceedings of International Conference on Parallel Processing (ICPP) 2,011: 155-164.

Harald Servat, Germán Llort, Judit Giménez, Kevin A. Huck, Jesús Labarta. “Folding: Detailed Analysis with Coarse Sampling” In proceedings of the Parallel Tools Workshop 2,011: 105-118.

Harald Servat, Germán Llort, Judit Giménez, Kevin A. Huck, Jesús Labarta. “Framework for a productive performance optimization**”** In Parallel Computing Journal, 39(8) 2,013: 336-353.

Harald Servat, Germán Llort, Juan González, Judit Giménez, Jesús Labarta. “Identifying code phases using piece-wise linear regressions” In proceedings of the 28th IEEE International Parallel & Distributed Processing Symposium (IPDPS) 2,014: 941-951.

Harald Servat, Germán Llort, Juan González, Judit Giménez, Jesús Labarta: “Bio-inspired call-stack reconstruction for performance analysis” Technical report UPC-DAC-RR-2014-20.