Visualization of Repetitive Patterns in Event Traces

Andreas Knüpfer, Bernhard Voigt, Wolfgang E. Nagel, Hartmut Mix

ZIH, TU Dresden, Germany
andreas.knuepfer@tu-dresden.de,
voigt@zhr.tu-dresden.de,
wolfgang.nagel@tu-dresden.de,
hartmut.mix@tu-dresden.de

Abstract. Performance Tracing has always been challenged by large amounts of trace data. Software tools for trace analysis and visualization successfully cope with ever growing trace sizes. Still, human perception is unable to "scale up" with the amounts of data.
With a new model of trace visualization, we try to provide *less data* but *additional information* resp. more convenient information to human users. By marking regular repetition patterns and hiding the inner details less complex visualization can offer better insight. At the same time irregular sections are revealed which are potentially interesting.
The paper introduces the origin of repetition patterns and outlines the detection algorithm used. It demonstrates the new visualization scheme which has also been incorporated into Vampir NG as a prototype. Finally, it gives an outlook on further development and possible extentions.

1 Performance Tracing

Event tracing is a well-established method for performance analysis of computer programs, especially in parallel and High Performance computing (HPC). It has a reputation for producing *large* amounts of trace data where "large" has always been defined by the time's standards.

Many advancements in HPC contributed to that. This includes faster processors and growing parallelism. Also, more detailed instrumentation and additional data sources increase trace data volume. Last but not least availability of larger memory and storage capacities made traces grow larger.

Therefore, trace analysis and visualization has always been a challenging task on contemporary computers and always will be. The more so as one cannot require a actual super-computer for analyzing super-computer traces.

But what is the effect from the human users' perspective? Could screen resolution grow with an appropriate rate? Can human perception *scale* with the growing amounts of data as well? And is there really *more information* when there is simply *more data*[1]? The next chapter tries to address those questions.

[1] Let "information" be what contributes to the users insight while "data" is just the byte sequence transporting information.

Then, Section 3 will propose a new visualization methodology to provide *more information* to the user with *less data*. The following Section will outline the algorithms for pattern detection. Finally, there is an outlook on improved analysis methods based on patterns.

2 Data vs. Information of Traces

There are several reasons for growing trace sizes. First, there may be larger software projects with bigger source codes. Also, it might grow larger due to optimization and specialization of code. Second, instrumentation and measurement evolve, providing additional data, for example performance counters in processors, in communication sub-systems and in I/O backends. Third, longer resp. faster running programs with more and more parallelism will increase the number of repeated executions of certain program parts.

The former two reasons have only secondary effect on growing trace data sizes. They will hardly increase it by orders of magnitude. Only the latter is responsible for traces of tens of gigabytes, today. On average the trace size will double when iteration count resp. run-time is doubled or when there are twice as many parallel processes.

In terms of information to the user the same situation looks different. Assumed it is appropriately described as "sequence $A = (a_1, a_2, \ldots)$ is repeated n times". From this point of view the statement "sequence A is repeated $k = 2n$ times" carries about the same amount of information but not twice as much.

There is some additional information concerning the single iterations, however. First, it is the regular standard behavior of related iterations. Most likely this is independent of iteration count. Second, there are outliers and abnormal cases differing from regular behavior.

This additional information is not accessible to the human user by pristine visualization. Rather sooner than later human perception will be overcharged by too much data. The aspect of regular or irregular behavior is in fact hidden.

Therefore, we want to propose a new scheme of visualization that reduces the amount of data in favor of real information perceivable by human users. Nevertheless, all familiar information *and* data will still be available through interactive control.

3 Visualization of Repetition Patterns

The new visualization approach focuses on Process Timeline Diagrams like found in the Vampir NG [3] tool, for example. For each and every function call a single box representing a state of execution, compare Figure 1a. Now, in a simplified Process Timeline Diagram all regular repetition patterns of arbitrary size and nesting depth are replaced with a single marked box. See Figure 1b for an example. Those boxes indicate a region with regular behavior, inner details are not shown.

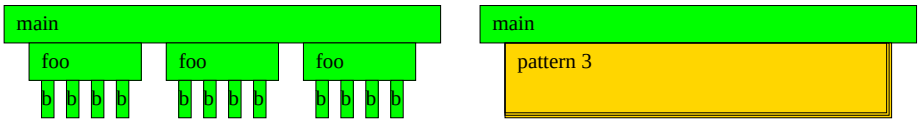

Fig. 1. Plain Process Timeline Diagram of some nested function calls (a), left, and simplified Pattern Diagram of the same situation (b), right. The box `pattern 3` states that there is a repetition of regular call sequences inside.

This display hides too much information. Therefore, patterns can be decomposed interactively. This replaces every instance of a pattern by its direct sub-patterns. See Figure 2 for an example. All sub-patterns can be decomposed as well until the fully decomposed view is identical to the traditional one (Figure 1a). In addition to decomposing patterns there is an info dialog available to provide the pattern structure as well as statistics about all occurrences.

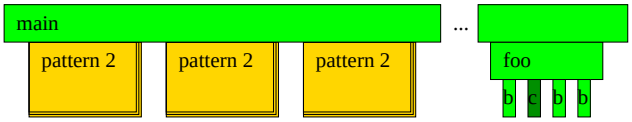

Fig. 2. Pattern Diagram after `pattern 3` in Figure 1b has been decomposed. The fourth call to `foo` is not covered by the pattern because there is one differing sub-call.

This view allows an easy distinction of regular an irregular parts of a trace. For example, the fourth iteration in Figure 2 is not covered by a pattern because there is a small structural difference. In general such important differences can hardly be perceived directly from a traditional timeline view. Either, because there are too many details to figure out, or because differences are not visible at all when there are more details than actual screen pixels.

4 Pattern Detection

The pattern detection algorithm is based on *Compressed Complete Call Graphs* (cCCGs) [1]. From this general purpose data structure it derives the so called *Pattern Graph* [2]. The cCCG references all call sequences that are regarded equal with respect to call structure *and* temporal behavior to a single representation. This allows notable data compression. See [1] for more details.

Now, the Pattern Graph summarizes this even more, neglecting any run-time information. Within all nodes it searches for repetition patterns of arbitrary length. To reduce computational overhead it regards only consecutive repetition patterns, but no patterns that appear isolated in multiple places. However, repeated patterns that appear in multiple places are identified as the same pattern.

With this constraint the complexity for pattern search is bounded by $O(n \cdot m)$ [2] where n is the number of nodes in the cCCG and m is the maximum number of regular child nodes per graph node. Because the cCCG's compression parameters are not important, here, the maximum compressed cCCG can be used, i.e. that with minimum node count n.

Pattern detection can also be performed in parallel such that every trace process/thread is handled separately. A final interchange of pattern information and renaming of local pattern identifiers is sufficient to produce global pattern information. Furthermore, it is possible to store pattern information persistently and reread it instead of re-computing it multiple times.

5 Outlook

The pattern detection and visualization outlined above have been incorporated into the Vampir NG tool as a prototype [2] see Figure 3.

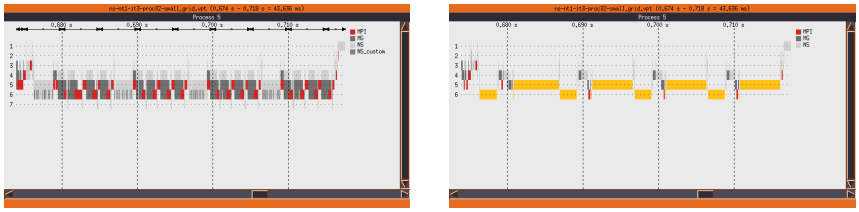

Fig. 3. Example screen shot of Vampir NG's [3] process timeline diagram in traditional way (a), left, and with pattern information (b), right.

So far, pattern detection and visualization relies on strutural identical patterns. Prospectively, pattern matching could accept certain variation. e.g. regard loops as equal if they have identical inner structure but different iteration counts. Furthermore, there should be automatic evaluation of the temporal behavior of all occurrences of the same pattern. Again, outliers from the standard behavior should be highlighted. Finally, the rendering of timeline diagrams with pattern information, which was derived from the traditional scheme, might be redesigned.

References

1. Andreas Knüpfer, Wolfgang E. Nagel: Compressible Memory Data Structures for Event-Based Trace Analysis. Future Generation Computer Systems, Volume 22, Issue 3, February 2006, pages 359–368
2. Bernhard Voigt: Effiziente Erkennungs- und Visualisierungsmethoden für hierarchische Trace-Informationen. Diploma thesis, TU Dresden, Germany, 2006
3. Holger Brunst, Wolfgang E. Nagel, Allen D. Malony: A Distributed Performance Analysis Architecture for Clusters. Proc. of IEEE International Conference on Cluster Computing, Hong Kong, China, IEEE Computer Society, pages 73–81, 2003