OJL によるトレースログ可視化ツールの開発

350702101　**後藤 隼弐**

要旨

近年，組込みシステムにおいても，マルチプロセッサの利用が進んでいる．その背景には，シングルプロセッサの高クロック化による性能向上効果の限界や，消費電力の増大がある．マルチプロセッサシステムでは，処理の並列性を高めることにより性能向上を実現するため，消費電力の増加を抑えることができる．しかし，マルチプロセッサ環境で動作するソフトウェアのデバッグは困難であるという問題がある．これは，処理の並列性からプログラムの挙動が非決定的になり，バグの再現が保証されないため，シングルプロセッサ環境で用いられているブレークポイントやステップ実行を用いた従来のデバッグ手法が有効でないからである．

一方，マルチプロセッサ環境で有効なデバッグ手法として，プログラム実行履歴であるトレースログを解析する手法が挙げられる．この手法が有効である理由は，並列プログラムのデバッグにおいて必要な情報である，各プロセスが，いつ，どのプロセッサで，どのように動作していたかということを，トレースログを解析することで知ることができるからである．しかしながら，開発者が直接トレースログを解析するのは効率が悪いという問題がある．これは，膨大な量となるトレースログから所望の情報を探し出すのが困難であることや，各プロセッサのログが時系列に分散して記録されるため，逐次的にログを解析することが困難であることが理由である．

トレースログの解析を支援する方法として，ツールによるトレースログの可視化が挙げられ，これまでに多くのトレースログ可視化ツールが開発されている．具体的には，組込みシステム向けデバッガソフトウェアや統合開発環境の一部，Unix 系 OS のトレースログプロファイラなどが存在する．しかしながら，これら既存のツールが扱うトレースログは，形式が標準化されておらず，環境（OS やデバッグハードウェア）毎に異なるため，可視化対象が限定されおり，汎用性に乏しい．さらに，可視化表示項目が提供されているものに限られ，追加や変更が容易ではないなど，拡張性に乏しいといった問題もある．

そこで我々は，これらの問題点を解決し，汎用性と拡張性を備えたトレースログ可視化ツールを開発することを目的とし，TraceLogVisualizer (TLV) を開発した．まず，TLV 内部でトレースログを抽象的に扱えるよう，トレースログを一般化した標準形式トレースログを定め，任意の形式のトレースログを標準形式トレースログに変換する仕組みを変換ルールとして形式化した．次に，トレースログの可視化表現を指示する仕組みを抽象化し，可視化ルールとして形式化した．TLV では，変換ルールと可視化ルールを外部ファイルとして与えることで，汎用性と拡張性を実現した．

開発した TLV を用いて，シングルコアプロセッサ用 RTOS(Real-time operating system) やマルチコアプロセッサ用 RTOS，組込みコンポーネントシステムなど，形式が異なる様々なトレースログの可視化を試み汎用性の確認を行った．また，可視化表示項目の変更，追加を試み拡張性の確認を行った．その結果，変換ルールと可視化ルールの変更，追加でこれらが実現可能であることを示した．

TLV の開発は，OJL(On the Job Learning) 形式で行い，開発プロセスに，ユースケース駆動アジャイル開発を適用して実施した．

Development of Visualization Tool for Trace Log by OJL

350702101 Junji Goto

Abstract

In recent years, multiprocessors have been used even in embedded systems. The reason for this is limits of effectiveness that improve a performance through overclocking in a single processor. A multiprocessor system can reduce the increase in power consumption even though improving performance by parallel processing. However, there is a problem that it is difficult to debug software executed in multiprocessors. This means that a traditional debugging way using breakpoints and step excuting is ineffective, because a repeatability of bug is made low by a nondeterministic behavior of parallel processing.

On the other hand, an effective technique to debug software excuted in multiprocessors includes analyzing a trace log that is a program execution history. The reason for that the technique is effective is that it can know necessary information for a parallel program debugging by analyzing the trace log. The information, for example, are when, where or how long processes executed. However, it is inefficient that developeres analyze the trace log directly, because searching a desired information in tremendous quantities of trace log and analyzing time-series trace log recorded by a number of processors sequentially are difficult.

Techniques to support developeres to analyze the trace log includes visualizing trace log by tools. And, many visualizing tools for trace log have been developed before now. In particular, there are debugging software or integrated development environment for embedded systems and trace log profilers for Unix-like operating systems. However, these existing tools lack general versatility because they treat only own format trace log. And, they lack expandability because visualized information are limited to items provided by them.

To that end, we developed TraceLogVisualizer(TLV), a visualization tool forthe trace log, for the purpose of implementation of general versatility and expandability. First, we defined the standard-format-trace-log with generalizing a trace log. This is necessity for that the TLV treat the trace log abstractively. And, we provided a mechanism that any format trace logs convert to the standard-format-trace-log by writing a convert-rule-file. Next, we formalized a mechanism associating shapes and trace logs as visualize-rule-file through abstracting them.

We confirmed general versatility through attemptting to visualize trace logs of a wide variety of formats including the trace log of RTOS (Real-time operating system) for a singlecore processor and multicore processors, and embedded component systems. Also, we confirmed expandability through attemptting to add and change visualized information. In the result, they are achieved by writing convert-rule-files and visualize-rule-files.

Development of TLV was performed as OJL(On the Job Learning), and development process carried out by applying use case driven agile development.