Performance Measurement Techniques

Principal Investigator
Dr. Christian Iwainsky

Project Term
2014 - 2014

Project Areas
Computer Science

Clusters
Lichtenberg Cluster Darmstadt

University
Technische Universität Darmstadt

Benchmark	Reference Runtime	Overhead
403.gcc	43,13 s	100,89%
429.mcf	266,72 s	61,48%
433.milc	495,16 s	13,35%
435.gromacs	594,15 s	7,97%
444.namd	499,42 s	7,49%
447.dealII	387,48 s	2087,03%
450.soplex	139,40 s	368,29%
453.povray	199,30 s	509,79%
456.hmmer	376,98 s	1,01%
458.sjeng	608,30 s	83,82%
462.libquantum	447,46 s	3,31%
464.h264ref	77,22 s	98,96%
470.lbm	458,81 s	0,19%
473.astar	156,48 s	193,13%
482.sphinx3	658,38 s	11,98%
DROPS	1,30 s	3195,75%

Introduction

In light of the complexity and diversity of current HPC platforms, programmers need to understand application execution behavior in order to identify performance bottlenecks and to support the compiler in generating efficient code. To this end, performance analysis tools use runtime measurements to provide information of the applications runtime behavior. Two established and widely used methods to gather such information are sampling or the instrumentation of the application. Each method has its advantages and disadvantages in terms of induced measurement overhead, measurement-control, data quality and the capability to map the data back to the source code.

Methods

Instrumentation is the modification of the application to induce measurement points, so called probes, in the application in order to gather the required information. The correlation to the source is implicit by the instrumentation location. These probes are either injected manually by the user, or automatically either by the compiler or a dedicated instrumentation tool. The runtime overhead for such an approach is composed of the overhead of all the probes (typically implemented by calls to a measurement library) and the frequency at which the individual probes are invoked at runtime. While the overhead of a single probe usually can be considered constant, the number of probe invocations is difficult to control when using current instrumentation technology. For example, C++ applications instrumented with current methods provided by tools like Scalasca or Vampir exhibit trillions of probe invocations during the applications runtime resulting in tremendous overheads (see Table). Summarizing, the conceptual overhead of instrumentation (Oinstr.) can be expressed as a product of the probes overhead (OProbe) and the number of probe invocations (ninvoc.): Oinstr.=OProbe*ninvoc. Sampling typically denotes the statistical gathering of relevant data during an applications execution by programming hardware interrupts (event based sampling, e.g. timer or executed instructions). In order to map the information back to the code, sampling tools read the program counter and analyze the call stack and the binary to derive the precise context for the sample, i.e. call path and source location. While mapping the sample to the source code relies on the debug information of the binary and can be obtained offline, the unwinding of the call stack must be done at runtime, as the context is potentially lost after the sampling event. As the precise structure of the call stack, that is its depth and composition, is unknown a priori, the tool has to unwind the call stack back to a known location, such as the main function. This unwinding is complex and costly. Therefore, several optimizations are applied to reduce the cost of unwinding, e.g. caching the lastanalyzed call stack state, adding markers to the call stack or unwinding only specific samples. However, the overhead of sampling remains dynamic and sample location dependent.

Results

Summarizing, the sampling overhead (Osampl.) is expressed as the sum of all the overheads of all the different sample locations O(loc.) times the number of samples taken at that location n(loc.): Osampl.= SUM(O(loc.)*n(loc.))

Using the Lichtenberg cluster, we evaluate and validate novel techniques to reduce incurred measurement overheads required for performance analysis whilst maintaining expressive data suitable for performance analysis.

JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN
TECHNISCHE UNIVERSITÄT DARMSTADT
UNIKASSEL VERSITÄT
Philipps Universität Marburg

Last Update: 2020-09-28 18:23

JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN
TECHNISCHE UNIVERSITÄT DARMSTADT
UNIKASSEL VERSITÄT
Philipps Universität Marburg