Scalable Parallel Debugging with Statistical Assertions

Minh Ngoc Dinh, David Abramson, Chao Jin
Monash University, Clayton, 3800, Victoria, Australia
{ngoc.minh.dinh,david.abramson,chao.jin}@monash.edu

Andrew Gontarek, Bob Moench, Luiz DeRose
Cray Inc, Saint Paul, MN 55101-2987, United States
{andrewg,rwm,ldr}@cray.com

Abstract

Traditional debuggers are of limited value for modern scientific codes that manipulate large complex data structures. This paper discusses a novel debug-time assertion, called a "Statistical Assertion", that allows a user to reason about large data structures, and the primitives are parallelised to provide an efficient solution. We present the design and implementation of statistical assertions, and illustrate the debugging technique with a molecular dynamics simulation. We evaluate the performance of the tool on a 12,000 cores Cray XE6.

Categories and Subject Descriptors **D.2.5** [Testing and Debugging**:]** Distributed debugging; Code inspections and walkthroughs

General Terms Performance, Verification.

Keywords parallel debugging, statistic, assertion

1. Introduction

Our earlier work has demonstrated that ad hoc debug-time assertions can assist in parallel debugging tasks because it is not necessary to examine every value in a large data structure [1]. More importantly, a parallel computer can be used to execute the assertion logic, making it efficient when used on large data structures and machines. This research introduces a new type of ad hoc debug-time assertion called a *Statistical Assertion*. Such assertions allow users to reason about derived metrics rather than the raw data. The essence of this approach is to (1) diminish the substantial amount of raw data to manageable "blocks"; (2) alleviate the complexity in managing the data decomposition across a large number of processors; and (3) leverage parallelism to make the system fast enough for real time debugging.

2. Statistical Assertions

A *statistical assertion* is defined as a user-defined predicate consisting of two *data models* in the form of either statistical primitives (e.g. mean or standard deviation values) or data models (e.g. histograms or density functions). Statistical assertions allow the user to compare data pattern information between two data structures, instead of comparing the exact values like the assertions used in our previous works [1, 2]. Our debugging framework addresses two significant issues. First, it needs to support a wide range of useful statistics, and it is desirable to compute these in parallel in order to provide real time debugging of large datasets. Second, the framework has to allow users to create arbitrary user-defined data models.

PPoPP'12, February 25–29, 2012, New Orleans, Louisiana, USA.
ACM 978-1-4503-1160-1/12/02.

2.1 Split-phase Statistical Operation

The parallel computation of basic statistics such as average, max, min is relatively straight forward; however, more complex statistics require special handling. The overall computation can be broken into two processes to make the parallel part explicit.

Parallel: calculate a set of primary statistics from the input dataset. This phase is embarrassingly parallel as it does not involving any inter-process communication; and

Aggregation: assemble a collection of primary statistics to form a full statistical model.

2.2 User-defined Abstract Data Model

As discussed earlier, users need to create arbitrary data models. In order to create accurate distributions, we need to perform quite a few computations, and it is desirable to parallelize this operation as well. The split-phase scheme presented above can be used to create distributions in parallel.

3. Implementation

3.1 User-defined Statistic Function

Users can create their own statistical reduction function using C programming language. The code is compiled and linked into the debugger binary in runtime. To make it easier for a user to write statistical functions, we have defined an API that standardises the interface. Furthermore, to enforce the split-phase statistic framework discussed in section 2.1, a general template is provided in which a few compulsory functions are expected.

```
func my_func(data)
    define my_func_server(data)=server_result
    define my_func_client(collection)=client_result
end
```

3.2 User-defined Data Models

To help users create abstract data models, we introduce a new debugger built-in function, called *randset*, that defines random variates. Currently, Guard supports various typical distribution models including Gaussian, Poisson, and Maxwell-Boltzmann.

```
randset(<distribution_name>,<dataset_size>,…)
```

3.3 Architectural Details

Guard employs a client/server model, where the debugger is divided into a single front-end client and multiple servers (Figure 2). The details of how this architecture is useful for processing ad hoc debug-time assertions is discussed fully in our previous works [1, 3]. More importantly, an assertion in Guard is compiled into a low-level graph description upon creation [3], and is executed by a special interpreter as illustrated in Figure 1.

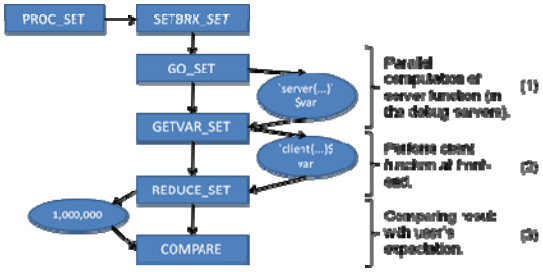

Figure 1 - Abstract execution of statistical assertion

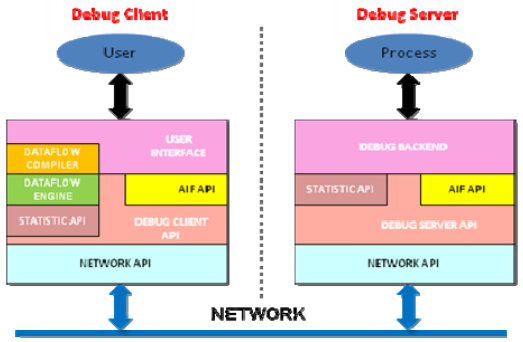

Figure 2 - Guard layered architecture with Statistic API

4. Case Study: Molecular dynamics

We used a parallel molecular dynamics code written in C using MPI and replicated a set of program bugs presented in Frenkel et. al. [4] to illustrate the expressive power and potential of statistical assertions for finding errors.

4.1 Monitoring Particles' Speeds

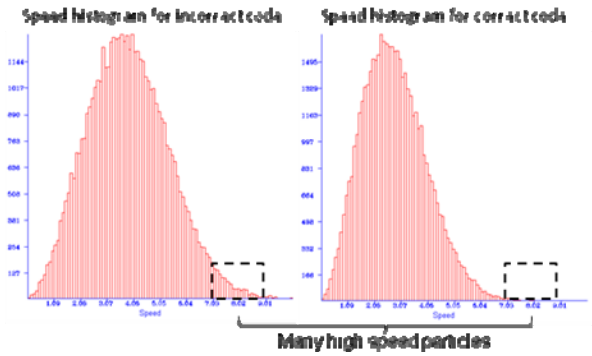

Figure 3 – Speed histogram comparison

In any given fluid, the speed varies a great deal, from very slow particles to very fast ones. However, this scalar value spreads according to Maxwell-Boltzmann distribution [4]. Therefore, monitoring particles' speeds can help detecting anomalies in a simulation. This is done with the *histogram assertion* below:

```
create $model=randset(maxwell, 49152, 6, 2)
set reduce histogram(100,0.0,10.0)
assert $a::speed_array@pmd.c:28~$model<0.02
```

4.2 Energy Conservation

The law of *energy conservation* states that the total amount of energy in an isolated system remains constant. Therefore, a drift of this quantity may signal programming errors. To detect this we trigger a *standard deviation assertion* after each simulation cycle to ensure this quantity does not alter significantly.

```
assert stdev($a::totEnergy@pmd.c:28) < 0.1
```

5. Performance Evaluation

We evaluate the performance on a Cray XE6 system, with 17,472 cores. To demonstrate that the approach is applicable to large-scale scientific codes, we use data structure sizes in a real molecular dynamics simulation with a 209×10^6 atoms [5].

5.1 Strong Scaling Experiment

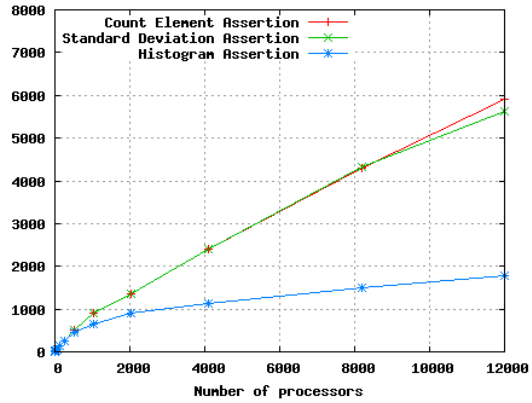

Figure 4 - Assertions speedup against #processors

5.2 Weak Scaling Experiment

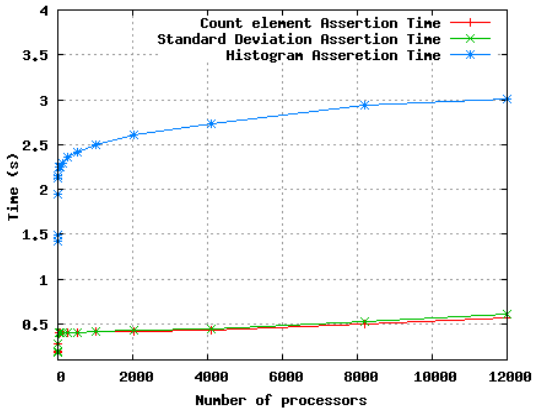

Figure 5 - Weak scaling assertion evaluation time

References

[1] M. N. Dinh, D. Abramson, D. Kurniawan, C. Jin, B. Moench, and L. DeRose, "Assertion based parallel debugging", in *CCGrid*, Newport Beach, 2011.

[2] D. Abramson, M. N. Dinh, D. Kurniawan, B. Moench, and L. DeRose, "Data Centric Highly Parallel Debugging", in *HPDC,* Chicago, 2010.

[3] G. R. Watson, "The Design and Implementation of a Parallel Relative Debugger", in *Faculty of Information Technology*. vol. PhD Thesis Melbourne: Monash University, 2000, p. 197.

[4] D. Frenkel and B. Smit, *Understanding Molecular Simulations: From Algorithms to Applications*, 2 ed. Elsevier Science & Technology, 2002.

[5] P. S. Branicio, R. K. Kalia, A. Nakano, and P. Vashishta, "Shock-Induced Structural Phase Transition, Plasticity, and Brittle Cracks in Aluminum Nitride Ceramic", *PHYSICAL REVIEW LETTERS,* vol. 96, issue 6, 2005.