ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №2(29), 2016, с. 127–135

УДК 004.051

Е. В. Петренко, И. Г. Черных, И. М. Куликов

Разработка системы анализа производительности приложений для мобильных платформ

Аннотация. Одним из наиболее важных этапов в процессе разработки программного обеспечения является анализ производительности приложений. В случае разработки и тестирования программных приложений для суперкомпьютеров анализ производительности как отдельных алгоритмов, так и приложений в целом, является ключевым.

В последнее время набирает популярность использование мобильных платформ для построения суперкомпьютеров благодаря низкому энергопотреблению систем и появлению 64-разрядных процессоров. В статье описывается система для запуска 32- и 64-разрядных тестов, созданная авторами, также приводятся результаты тестирования на характерном для мобильных платформ наборе задач, выбранном исходя из анализа частоты использования приложений пользователями устройств на мобильных платформах.

Ключевые слова и фразы: Мобильные платформы, суперкомпьютеры, эффективность, анализ производительности.

Введение

Анализ производительности при разработке приложений занимает важное место наряду с модульным и многими другими видами тестирования. При разработке приложений и архитектур для суперкомпьютеров подобное тестирование оказывается еще более важным ввиду высокой стоимости эксплуатации такого рода систем. В работе предложена кроссплатформенная расширяемая система анализа производительности приложений со встроенным набором тестовых сценариев. Данная система предоставляет возможность запускать тесты на разных платформах и получать результат в виде операций в секунду (микросекунду, наносекунду). Также она позволяет создавать собственные сценарии и выбирать, какие именно операции считать.

Работа была частично поддержана грантом РФФИ №15-31-20150_мол_а_вед.

В статье представлен экспериментальный образец в виде приложения под платформу Android с удобным графическим интерфейсом. Система была разработана на языке Java, в дальнейшем планируется опубликовать ее с открытым исходным кодом. В качестве тестов производительности использовались: алгоритмы с интенсивным использованием длинной арифметики [1,2], метод Рунге-Кутты [3, 4], JNI, средства шифрования из криптопровайдера BouncyCastle [5, 6], стандартная библиотека Java для сжатия [7], BoofCV, Jbox2D, jbullet, libGDX. Система позволяет запускать тесты и получать результат как на мобильном устройстве, так и на любой другой платформе, для которой существует виртуальная машина Java.

1. Постановка задачи. Требования к системе

После детального анализа задачи к системе предоставлялись следующие требования:

Открытый исходный код, нужный для воспроизводимости результатов.

Стабильность результатов, важная для сравнения платформ.

Легкая настройка, предполагающая создание файла конфигурации для запуска тестов с разными параметрами.

Детальная статистика по результатам.

Запуск во множестве режимах, позволяющий использовать разные автоматизированные технологии тестирования, например, скрипты запуска.

Простая проверка адекватности результатов, обеспечивающая осмысленность анализа.

Графический пользовательский интерфейс, облегчающий использование инструмента.

Для начала была предпринята попытка использовать существующий инструмент и проведено исследование следующих систем тестирования:

TradeFed — изначально код сделан под Android, но сбор статистики производительности был практически невозможен.

Caliper — инструмент компании Google, прекрасно подходит для тестирования производительности, но не имеет графического интерфейса и очень плохо приспособлен для запуска на Android.

В результате было принято решение создать собственный инструмент. За основу был взят инструмент под названием MTTest, разработанный компанией Intel на языке C и выложенный с открытым исходным кодом. Затем идея MTTest была реализована на языке Java.

2. Методы решения

Система для своей работы использует набор микротестов и конфигурационные файлы в формате XML, содержащие настройки каждого теста (количество повторений, количество потоков, параметры вывода результатов, также индивидуальные настройки).

Общая структура тестового сценария:

Java

```
class MyBenchmark extends AbstractTest {
   // various fields
   ...

   // specific fields-parameters
   protected String paramGoldenfile; //name of input file
   ...

   @Override
   public void init(Config config)
     throws InvalidTestFormatException;

   @Override
   public void iteration()
     throws TestRuntimeErrorException;

   @Override
   public void done();

   // various methods
   ...

}
```

Тестовый сценарий пишется в виде класса, наследуемого от специального класса самой системы AbstractTest. Поля класса, имена которых начинаются с param, являются настройками теста и могут быть заданы извне. Затем реализуются методы init, iteration, done, которые, исходя из названия, предназначены соответственно для инициализации тестовых данных, подсчета количества операций и действий по завершении измерений.

Метод iteration в общем виде можно записать следующим образом:

Java

```
@Override
public long iteration()
      throws TestRuntimeErrorException {
   long count = 0;
   for(int i = 0; i < paramReps; i++){
      // pay load operations
      if(params.isValidating){
          // test specific check of output
      }
   }
   return count;
}
```

Итерация теста — это множество внутренних итераций интересующих нас операций, в конце цикла необходимо делать count += n, где n — количество операций, которые нас интересуют. После своего исполнения этот метод возвращает количество совершенных операций, которые сопоставляются со временем исполнения метода и на основе полученных данных вычисляется результат — количество операций в секунду (в настройках можно секунды заменить на миллисекунды или наносекунды).

Каждый тест запускается в указанном количестве потоков, исполнение проходит в 3 этапа:

Разогрев — стадия, на которой проводится вывод процессора из режима экономии электроэнергии (по необходимости), инициализация виртуальной машины, работа JIT-компилятора.

Измерения — производится подсчет операций.

Заключение — это финальная фаза, когда поток еще работает над исполнением теста, но ничего не измеряет, гарантирующая сохранение нагрузки до окончания измерений всеми другими потоками.

Указанный способ исполнения освобождает от ненужных синхронизаций между потоками.

В настройках можно задавать параметры минимальной длительности каждой фазы. Фактическая длительность итерации теста определяется трудоёмкостью операций и количеством повторений paramReps, но тест повторяется до истечения заданного в настройках времени.

Предусмотрена возможность проверки адекватности результатов. Например, может выдаваться предупреждение недостаточной фактической длительности итерации (меньше 200 мс). Это необходимо как для минимизации накладных расходов на запуск теста, так и для гарантии неизменности нагрузки в случаях, когда фактическая длительность итерации превышает длительность остывания.

Проверке поддается также и логика теста. Такая проверка пишется отдельно для каждого теста и позволяет убедиться, что тестовый сценарий написан правильно, и выходные данные соответствуют входным. После этого проверки можно отключить в XML-файле, чтобы исключить их влияние на результаты измерений.

Конфигурационный файл пишется следующим образом:

Config

```
<mttest>

    <!--Global configuration -->
    <conf name=‘‘globalOpt’’ value=‘‘val1’’ />

    <!--Benchmark invocation -->
    <benchmark name=‘‘className1’’>
       <!--Per benchmark configuration -->
       <option name=‘‘localOpt’’ value=‘‘val2’’/>
    </benchmark>

    <benchmark name=‘‘className2’’>
      <option name=‘‘otherLocalOpt’’ value=‘‘val2’’/>
    </benchmark>

</mttest>
```

3. Численные результаты

Всего для данной системы разработано порядка 100 тестов на разные области, из них 40 уникальные[1]. Тесты группированы по природе выполняемых операций, которые создают нагрузку для тех или иных блоков процессора (регистры, арифметические блоки и т.д.). При создании тестов учитывались в первую очередь различия архитектур 32 и 64 бита [8]. Так, в качестве примера, можно привести увеличенную разрядность регистров, что позволяет не разделять числа типа

[1]Под уникальным тестом подразумевается тестовый сценарий, а практически сам тест дублирован на разные типы данных и размерности

Таблица 1. Результаты выполнения тестов

Область	32 бита, оп./сек.	64 бита, оп./сек.	Ускорение
Сортировка	3322073	3606652	8,56%
Математика	99713394	144153902	44,55%
Рунге-Кутта	21937984	29046069	32,4%
libGDXAI	3150	125722	3891%
BoofCV	30,56	43,24	41,4%
Сжатие	168,1	233,0	38,59%
Шифрование	4,558	6,44	41,36%
Физика	22,15	26,68	20,45%
JNI	18452	18497	0,24%
Общий результат	9553	15729	64,6%

long при хранении на 2 регистра, а хранить все в одном, что дает большие преимущества в скорости [8, 9]. Результаты представлены в таблице 1. В столбце «Область» представлена группировка тестов по типичным задачам, каждая из них включает от 5 до 20 тестов. В столбцах 32 и 64 бита приведены средние геометрические результатов измерений производительности в каждом тесте соответствующей группы. Так как для каждого теста количество операций в секунду может отличаться на много порядков, среднее арифметическое плохо годится на роль среднего. Внизу таблицы вычислено среднее геометрическое во всем представленным областям, а в крайнем правом столбце указано относительное ускорение.

Профилирование и детальный анализ и интерпретация результатов еще не проводился, поэтому имеют место некоторые неадекватные данные, например, система выдала, что библиотека libGDX для моделирования искусственного интеллекта в играх дает прирост на 64 битах до 3891%. Эти ситуации подлежат детальному анализу.

Заключение

С помощью разработанной авторами системы можно, используя встроенный набор тестов, измерить и сравнить производительность устройства, созданного на мобильной платформе, с другими. Запустить тестирование можно на любой платформе, для которой реализована Java-машина. Графический интерфейс делает процесс гораздо

более наглядным. Практическая часть работы реализована на языке Java и включает в себя следующие компоненты:

Инструментарий для запуска тестов производительности.
Графическая среда для настройки окружения и запуска тестов на ОС Android.
Набор скриптов на языке bash и конфигурационных файлов для запуска тестов в ОС Linux.
Набор готовых тестов для анализа преимуществ архитектуры Intel x86 64 бита над x86 32 бита.

На данном этапе проведен анализ скорости исполнения на платформе Android на процессоре Intel Atom 32 бита и 64 бита [8]. На основе тестирования были даны рекомендации производителю аппаратно-программной платформы для повышения производительности программных компонентов.

Список литературы

[1] Ш. Т. Ишмухаметов, *Методы факторизации натуральных чисел*, ИМ СО РАН, Новосибирск, 1999, с. 36. ↑

[2] D. Knuth, *The Art of Computer Programming*, Ch. 4.4. V. 2: *Seminumerical Algorithms*, Third Edition, Addison-Wesley, Reading, Massachusetts, 1997, xiv+762 p. ↑

[3] Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков, *Численные методы*, Бином, М., 2001, с. 363—375. ↑ 128

[4] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II*, Springer Series in Computational Mathematics, vol. **14**, Springer, Berlin–Heidelberg, 1996, pp. 40–41. ↑ 128

[5] А. Ю. Нестеренко, *Введение в современную криптографию. Теоретико-числовые алгоритмы*, Гос. ин-т. электроники и математики, М., 2012, с. 43–69. ↑ 128

[6] Й. Ян Сонг. *Криптоанализ RSA*, ИМ СО РАН, Новосибирск, 1999. ↑ 128

[7] Ф. А. Новиков, *Дискретная математика для программистов*, Питер, СПб, 2001, с. 165–187. ↑ 128

[8] J. Mashey. “The Long Road to 64 Bits”, *ACM Queue*, **4**:8 (2006), pp. 85–94. ↑ 131,132,133

[9] *i860 Processor Family Programmer’s Reference Manual*, Pdf, Intel, 1991 (Retrieved October 7, 2013). ↑ 132

Рекомендовал к публикации *Программный комитет*
Четвёртого национального суперкомпьютерного форума ***НСКФ-2015***

Пример ссылки на эту публикацию:

Е. В. Петренко, И. Г. Черных, И. М. Куликов. «Разработка системы анализа производительности приложений для мобильных платформ», *Программные системы: теория и приложения*, 2016, **7**:2(29), с. 127–135.

URL: http://psta.psiras.ru/read/psta2016_2_127-135.pdf

Об авторах:

Евгений Викторович Петренко

Студент-магистрант 2 курса ФИТ НГУ. Научные интересы: анализ данных, суперкомпьютерные вычисления, вычисления на графических процессорах, машинное обучение

e-mail: mvEvgenX@hotmail.com

Игорь Геннадьевич Черных

Окончил Новосибирский Государственный Университет в 2002г., кандидат физико-математических наук. Старший научный сотрудник ИВМиМГ СО РАН, уч. секретарь ЦКП Сибирский Суперкомпьютерный Центр ИВМиМГ СО РАН. Область научных интересов: суперкомпьютерные вычисления, астрофизика, химическая кинетика.

e-mail: chernykh@parbz.sscc.ru

Игорь Михайлович Куликов

Окончил Новосибирский Государственный Технический Университет, кандидат физико-математических наук. Область научных интересов: вычислительная астрофизика, космология, суперкомпьютерные вычисления.

e-mail: i.kulikov@corp.nstu.ru

Eugeniy Petrenko, Igor' Chernykh, Igor' Kulikov. *Development of a system for performance analysis of mobile applications.*

Abstract. In software development performance analysis is one of the most important thing, like unit testing and other more. While you develop an application or architecture for supercomputer system, the importance of accurate analysis is raising due to high usage costs of supercomputer.

In progress of work, which the article is talking about, a framework for starting benchmarks is developed and now shows good results for mobile applications.

Benchmarks can be run on huge set of platforms, efficient multithreading is implemented, also you can send real data to a test and get the result of operations.

In this article describes an approach to benchmarking and expose ideas to improve the framework to use in scientific applications and supercomputers. (*In Russian*).

Key words and phrases: Mobile platforms, supercomputers, efficiency, performance analysis.

References

[1] Sh. T. Ishmukhametov, *Methods of integer factorization*, IM SO RAN, Novosibirsk, 1999, pp. 36 (in Russian).

[2] D. Knuth, *The Art of Computer Programming*, Ch. 4.4. V. 2: *Seminumerical Algorithms*, Third Edition, Addison-Wesley, Reading, Massachusetts, 1997, xiv+762 p.

[3] N. S. Bakhvalov, N. P. Zhidkov, G. M. Kobel'kov, *Numerical methods*, Binom, M., 2001, pp. 363—375 (in Russian).

[4] E. Hairer, G. Wanner, *Solving Ordinary Differential Equations II*, Springer Series in Computational Mathematics, vol. **14**, Springer, Berlin–Heidelberg, 1996, pp. 40–41.

[5] A. Yu. Nesterenko, *Introduction to modern cryptography. Number-theoretic algorithms*, Gos. in-t. elektroniki i matematiki, M., 2012, pp. 43–69 (in Russian).

[6] Y. Yan Song. *Cryptanalysis of RSA*, IM SO RAN, Novosibirsk, 1999 (in Russian).

[7] F. A. Novikov, *Discrete mathematics for computer programmers*, Piter, SPb, 2001, pp. 165–187 (in Russian).

[8] J. Mashey. "The Long Road to 64 Bits", *ACM Queue*, **4**:8 (2006), pp. 85–94.

[9] *i860 Processor Family Programmer's Reference Manual*, Pdf, Intel, 1991 (Retrieved October 7, 2013).

Sample citation of this publication:

Eugeniy Petrenko, Igor' Chernykh, Igor' Kulikov. "Development of a system for performance analysis of mobile applications", *Program systems: theory and applications*, 2016, **7**:2(29), pp. 127–135. (*In Russian*).

URL: http://psta.psiras.ru/read/psta2016_2_127-135.pdf