ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 3(17), 2013, с.??—??

УДК 004.051

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

Разработка системы анализа производительности приложений для мобильных платформ[1]

Аннотация. 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 последнее время набирает популярность использование мобильных платформ для построения суперкомпьютеров благодаря низкому энергопотреблению систем и появлению 64 разрядных процессоров. В ходе работы, которой посвящена данная статья, разработана система для запуска тестов, которая на данный момент отлично показала себя для мобильных платформ. Тесты можно запустить на большом количестве платформ, в системе эффективно реализуется многопоточность, а также можно подавать реальные данные и получать результат, как на компьютере, так и на мобильном устройстве. В статье описывается подход к тестированию производительности, а также идея развить этот инструмент в сторону применения для научных приложений и суперкомпьютеров.

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

Введение

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, ка-

[1] (Рекомендована к публикации.... Поддержана...!)

кие именно операции считать. 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 [6,7], стандартная библиотека Java для сжатия [5], 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:

(1) Открытый исходный код
(2) Стабильность результатов
(3) Легкая настройка
(4) Детальная статистика по результатам
(5) Запуск во множестве режимах
(6) Простая проверка адекватности результатов
(7) Графический пользовательский интерфейс

Открытость исходного кода нужна для проверки результатов. Стабильность результатов очень важна для сравнения платформ. Легкая настройка подразумевает под собой возможность создания файла конфигурации для запуска тестов с разными параметрами. Запуск во множестве режимов позволяет использовать разные автоматизированные технологии тестирования, например, скрипты запуска. Тест должен делать осмысленные вещи, поэтому проверка адекватности результатов должна быть непременно. А графический интерфейс создает легкость в использовании инструмента.

Для начала была предпринята попытка использовать существующий инструмент. Были исследованы следующие системы тестирования:

- TradeFed - изначально код сделан под Android, но сбор статистики производительности был практически невозможен.
- Caliper - инструмент компании Google, прекрасно создан для тестирования производительности, но не имеет графического интерфейса и очень плохо приспособлен для запуска на Android.

Таким образом, было принято решение написать 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, такие как: количество повторений, количество потоков, параметры вывода результатов, также индивидуальные настройки.

Общая структура тестового сценария:

```
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:

```
@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 - операции в секунду. Можно в настройках изменить на миллисекунды или наносекунды.

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

- Разогрев – стадия, на котором проводится вывод процессора из режима экономии электроэнергии, если такое было, инициализация виртуальной машины, работа JIT-компилятора.
- Измерения – собственно, производится подсчет операций.
- Заключение – это финальная фаза, когда поток еще работает над исполнением теста, но ничего не измеряет, которая будет гарантировать, что будет сохраняться нагрузка, пока все другие потоки не закончили измерения. Тем самым, это освобождает от ненужных синхронизаций между потоками.

В настройках можно задавать параметры минимальной длительности каждой фазы. Примечание о словосочетании "минимальная длительность": фактическая длительность итерации может быть неограниченно больше этого параметра, потому что если операции очень трудоемки или paramReps выставлен большим - то и время исполнения превысит значения параметра минимальной длительности. В свою очередь, если длительность будет меньше указанного параметра - итерация повторится, пока не проработает минимальную длительность.

Предусмотрена возможность проверки адекватности результатов, например, может выдаваться предупреждение недостаточной фактической длительности итерации (меньше 200мс), это нужно для того, чтоб минимизировать накладные расходы на запуск теста. Или же если значение длительности остывания меньше, чем фактическая длительность итерации, потому что в этом случае не будет гарантироваться постоянная нагрузка.

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:

```
<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, под уникальным тестом подразумевается тестовый сценарий, а практически сам тест дублирован на разные типы данных и размерности. 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 бит [9]. Так, в качестве при-

мера, можно привести увеличенную разрядность регистров, что позволяет не разделять числа типа long при хранении на 2 регистра, а хранить все в одном, что дает большие преимущества в скорости [8, 9]. Результаты представлены в таблице 1.

Таблица 1. Результаты выполнения тестов.

Область	32-бит, оп./сек.	64 бита, оп./сек.	Ускорение
Сортировка	3322073,555	3606652,042	**8,56%**
Математика	99713394,13	144153902,1	**44,55%**
Рунге-Кутта	21937984,3	29046068,92	**32,4%**
libGDXAI	3149,740149	125721,5104	**3891%**
BoofCV	30,573351	43,23572	**41,4%**
Сжатие	168,126149	233,01281	**38,59%**
Шифрование	4,558241	6,443914	**41,36%**
Физика	22,148413	26,678558	**20,45%**
JNI	18452,41575	18497,35677	**0,24%**
Общий результат	9552,797151	15728,94801	**64,6%**

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 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-бит над 32-бит.

На данном этапе проведен анализ скорости исполнения на платформе Android на процессоре Intel Atom 32-бита и 64-бита [9]. На основе тестирования были даны рекомендации производителю аппаратно-программной платформы для повышения производительности программных компонентов.

Благодарности. *Авторы благодарны компании Intel за поддержку проекта, работа поддержана грантом РФФИ (15-31-20150 мол-а-вед).*

Список литературы

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

[2] Donald E. Knuth «The Art of Computer Programming», том 2, Секция 4.4

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

[4] Hairer & Wanner 1996, pp. 40–41

[5] Новиков Ф.А. Дискретная математика для программистов // С-Пб: Питер, 2001, с. 165–187.
[6] Нестеренко А.Ю. Введение в современную криптографию. Теоретико-числовые алгоритмы // Москва: гос. ин-т. электроники и математики, 2012, с. 43–69.
[7] Ян Сонг Й. Криптоанализ RSA // Новосибирск: ИМ СО РАН, 1999.
[8] i860 Processor Family Programmer's Reference Manual (PDF). // Intel. 1991. Retrieved October 7, 2013.
[9] Mashey, John (October 2006). "The Long Road to 64 Bits" // ACM Queue 4 (8): 85–94. Retrieved 2011-02-19.

Об авторе:

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.

Игорь Михайлович Куликов

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

Образец ссылки на публикацию:

Е. В. Петренко, И.Г. Черных, И.М. Куликов. Разработка кросплатформенной графической системы запуска тестов производительности // Программные системы: теория и приложения: электрон. научн. журн. 2015. Т. 4, № 3(17), с. ??–??.
URL: http://psta.psiras.ru/read/???

E.V. Petrenko, I.G. Chernykh, I.M. 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.

Key Words and Phrases: Mobile platforms, supercomputers, efficiency, performance analysis.