УДК 681.03.06

В.В. Хашковский, В.Н. Лутай, В.В. Юрченко

СРАВНИТЕЛЬНАЯ ОЦЕНКА ВРЕМЕНИ ВЫПОЛНЕНИЯ ПРОГРАММ НА РАЗЛИЧНЫХ ПЛАТФОРМАХ

В работе рассматривается технология точного измерения времени выполнения функций для приложений, реализуемых на различных операционных системах и платформах. Для сравнения выбраны ОС Windows XP и Linux, а в качестве платформ языки C и Java, создающие приложения в виде бинарного и байт-кода соответственно. Обосновывается инструментарий, используемый при проведении измерений, производится сравнение различных методов измерения времени, основанных как на специфичных для ОС средствах, так и на платформонезависимых и аппаратно-специализированных. Приводятся результаты измерений.

Языки C и Java; нативный код;байт-код; библиотечные вызовы функций; внутренние вызовы функций; технология измерений времени выполнения функций; счетчики производительности.

V.V. Khashkovsky, V.N. Lutay, V.V. Yurchenko

COMPARATIVE ESTIMATION OF EXECUTION TIME OF PROGRAMS ON VARIOS PLATFORMS

In this paper we study the technology of exact measurement of execution time of functions for the applications realised on various operating systems and platforms. For matching are selected OS Windows XP and Linux and languages C and Java, creating applications in the form of binary and a byte-code accordingly. We consider the toolkit used at carrying out of various methods of time measurement, grounded both on resources specific to OS, and on platform-independent and hardware-specialised methods. Results of measurements are presented.

Languages C and Java; native code; byte-code; library call function; internal call function; methods of time measurement toolkit; counter of productivity.

Наиболее широкое распространение в настоящее время получили операционные системы Windows и Linux. Именно между ними, как правило, выбирают администраторы информационных систем и разработчики программного обеспечения. Выбор операционной системы во многом определяет и средства разработки, которые применяются для разработки программного обеспечения. И, если с разработкой системного ПО в большинстве случаев выбор вполне определен – обычно это язык C или его специализированные для различных платформ диалекты, то при выборе средств разработки для прикладного ПО анализируются два класса средств разработки: средства, генерирующие так называемый нативный (native) код и средства, генерирующие байт-код (машинно-независимый код низкого уровня). Класс нативных систем более широк и занимает изрядную долю всех систем. Отличается этот класс тем, что в результате обработки (компиляции и кодогенерации) исходного текста получается исполнимый образ программы, который содержит инструкции, «понимаемые» непосредственно центральным процессором. В отличие от нативных средства, ориентированные на генерацию байт-кода, генерируют машинно-независимый код низкого уровня, исполняемый интерпретатором, или, как часто говорят, виртуальной машиной. В этом плане наиболее широко известна Java, а также набирающий популярность C#. Отличительной особенностью байт-кода является его непосредственная переносимость для различных платформ. Для нативного кода переносимость (или, как часто говорят, портируемость) обеспечить более сложно; в любом случае требуется, как компилятор для

целевой платформы, так и соответствие операционной системы целевой платформы определенным требованиям, например, POSIX.

С другой стороны, традиционно считается, что производительность байт-кода (хотя он и проходит этап предварительной компиляции) ниже, чем производительность нативного кода. И такое распространенное мнение часто ограничивает разработчиков систем, требующих повышенной производительности, средствами только нативной кодогенерации. А поскольку для таких систем, как правило, используются различного рода Linux/Unix системы, то, в подавляющем большинстве случаев, для разработки высокопроизводительных приложений выбор падает на язык программирования C. Однако разработка современных систем представляет собой, как правило, коллективный труд и соответствующие технологии, ориентированные на организацию совместного производства программного продукта; при этом оказываются задействованы специфические возможности целевой ОС, что делает портируемость большой программной системы практически невозможной, в отличие от изначально кроссплатформенной разработки, например, на Java.

Очевидно, что выбор операционной системы и платформы для разработки приложений может быть сделан на основе данных о времени выполнения тех или иных операций. Задача измерения времени выполнения возникает в связи с организацией профилирования приложения собственными силами без привлечения специализированных инструментов от сторонних производителей.

В работе обосновывается методика проведения экспериментов по определению времени вызова методов и/или функций в программах на C и Java и приводятся их результаты. При этом производится сравнение различных методов измерения времени, основанных как на специфичных для ОС средствах, так и на платформонезависимых и аппаратно-специализированных.

Как известно, используемые в программном коде на C вызовы могут быть внутренними и библиотечными (системные вызовы здесь не рассматриваются). Для внутренних вызовов характерно их статичное нахождение в сегменте кода; время внутренних вызовов не будет увеличено из-за необходимости подгрузки динамической библиотеки. Для библиотечных вызовов возможны также два варианта подключения библиотеки: статическую на этапе сборки и динамическую загрузку в адресное пространство прикладного процесса посредством специального системного вызова. Соответственно и вызов функции, находящейся в библиотеке, по времени может отличаться от вызова статической функции.

Для Java также существует определенная специфика вызова методов, связанная с организацией подгрузки кода Java-машиной (JVM – Java Virtual Machine) по мере запроса со стороны приложения, а также возможностью обращаться из Java-программы к библиотечным функциям, написанным, например, на C и скомпонованным в виде динамически загружаемых библиотек. Эта возможность имеет исключительную важность для всего исследования в целом, поскольку предоставляет возможность проведения схожих по методу и технологии временных замеров, а также позволяет адекватно сопоставить полученные временные замеры. Кроме того, непосредственно выполнение измерения времени также вносит определенную задержку, которая необходимо измерить и учитывать при проведении основных испытаний.

В зависимости от программно-аппаратной платформы, языка программирования и API операционной системы возможно использование различных методов измерения, отличающихся по точности. Так для Java вне зависимости от платформы (что и является отличительной чертой байт-кода) существует возможность использования методов, предоставляемых JVM: *System.currentTimeMillis()*, *System.nanoTime()*, *Perf.getPerf().highResCounter()*, возвращающих время в миллисекундах и наносекундах соответственно. Для программ на C имеются различные

возможности в среде Windows и Linux/Unix. Так в среде Windows доступны так называемые счетчики производительности – *QueryPerfomanceCounter* и *QueryPerfomanceFrequency*, однако для Linux не имеется аналогичных счетчиков такого разрешения (секундное разрешение недопустимо грубо в условиях многозадачной системы с длительностью кванта 20-50 мс.). Вне зависимости от операционной системы возможно применение метода, основанного на использовании специализированного регистра центрального процессора, который является счетчиком тактов процессора с момента запуска – RDTSC (Read Timestamp Counter). Использование RDTSC позволяет, в частности, абстрагироваться от "наносекунд" и, следовательно, от конкретных частот работы оборудования. Используя значения RDTSC и зная частоту работы процессора и системной шины, не составляет труда перейти к "секундам" и их производным. Используя же одновременно различные методы оценки производительности, можно оценить их точность относительно наиболее "стабильного" показателя – счетчика RDTSC.

Для определения коэффициента перехода от RDTSC к секундам на практике хороший по точности результат показал именно таймер низкого разрешения. Вычисление этого переходного коэффициента получено при помощи определения количества тактов за фиксированный промежуток времени – 1 секунда. В ходе работы была также установлена точность описанного подхода для определения переходного коэффициента, относительная погрешность которого составила 0,00002%, что на частоте работы процессора 2 GHz составляет 40000 тактов (примерно 20000 нс.).

Особенностью проводимых экспериментов является также и то, что использование счетчика RDTSC из нативного кода не составляет труда – достаточно применить ассемблерные вставки для обращения к конкретным регистрам процессора, а для интерпретируемого кода это, естественно, невозможно. Поэтому для реализации аналогичного механизма на Java был задействован специализированный подход на основе JNI (*Java Native Interface*), который позволяет обращаться выполнять вызов функций, тела которых находятся в динамически подгружаемых библиотеках, написанных на C.

Для проведения экспериментального исследования применялись два разных способа: *Внешнее* и *Внутреннее* измерение. Особенность метода «*Внешнее измерение*» заключается в том, что измерение времени производится до вызова функции и после. В случае метода «*Внутреннее измерение*» применяется другой подход к измерению. Функция, которая возвращает временные единицы на момент ее вызова, относительно начала работы процессора либо относительно 00:00 01.01.1970 г., помещена в тело библиотечной функции, которая, в свою очередь, возвращала, полученные временные единицы. Для измерения работы такой функции необходимо произвести два вызова. Полученные результаты характеризуют промежуток времени от возврата из функции при первом вызове, до возврата при втором. В этот промежуток входит возврат из функции и вхождение в нее. Поскольку штатные обертки входа и выхода из функции, генерируемые компилятором одинаковы в обоих случаях, то теоретически и временные замеры, полученные обоими способами, должны быть эквивалентны. Однако потенциальные возможности внутреннего измерения значительно шире, а именно для Java без ограничения общности экспериментальной базы, внешняя функция может быть размещена в динамической библиотеке, доступной через JNI. Это дает возможность снять результаты измерения с максимальным разрешением – в тактах процессора.

При этом все эти результаты являются «грязным» замером, так как полученное время содержит в себе время работы самого измерения. Поэтому для чистоты эксперимента необходимо определить время работы самого измерения и полученное значение учитывать при расчете «чистого» времени выполнения кода.

Таким образом, в исследовании выполнялись следующие замеры:

- EF (Empty function) – время измерения пустого метода (функции);
- EM (Empty measurement) – время работы самого измерения;
- Время измерения работы библиотечного метода
  - ExtM (External measurement) – внешнее измерение;
  - IntM (Internal measurement) – внутреннее измерение.

Полученные в результате эксперимента данные представлены в таблице.

Таблица

Данные измерений

ОС	ЯПВУ	Метод измерения	Время по технологии измерения (нс)			
			EF	EM	ExtM	IntM
Linux Open Suse 11, AMD 2 ГГц (SLES, Intel Xeon 2,33ГГц QuadCore)	Java	RDTSC	n/a	n/a	n/a	108 (95,7)
		NanoTime	998 (399)	926 (372)	1032 (419)	n/a
		PerfCouter	1000 (0)	1000 (0)	2000 (0)	n/a
	C	RDTSC	38 (133,1)	36 (130,2)	39 (135,1)	41 (165,4)
Windows XP SP2	Java	RDTSC	n/a	n/a	n/a	155
		nanoTime	2104	1992	2114	n/a
		PerfCouter	2234	2514	2793	2793
	C *Static Link*	RDTSC	39	38	40	41
		PerfCouter	2128	2117	1906	1956
	C *Dynamic Link*	RDTSC	39	38	45	48
		PerfCouter	2128	2117	4638	2797

Результаты, полученные в ходе исследования, показывают, что использование счетчика тактов предоставляет более точные результаты, чем результаты, полученные при вызове системных API функций. Интересно сопоставление для Java аналогичных измерений и производительности компьютеров: на машине AMD 2 ГГц внутреннее измерение заняло 108 нс., а на машине Xeon 2,33 ГГц – 95,7 нс. В данном случае количество тактов для первого случая составит 2*108=216, а для второго – 2,33*95,7=223. Погрешность не более 3% на наносекундном базисе. ОС Linux обеспечивает несколько большее быстродействие, нежели ОС Windows. Использование штатных механизмов измерения времени в Java сопряжено с большой погрешностью измерения. Вопрос, связанный с возможностью использования Java для разработки высокопроизводительных приложений имеет, по всей видимости, двоякий ответ. С одной стороны, время простого вызова библиотечного метода Int.M. (для которого можно сравнить RDTSC) – отличается примерно в 2,5 раза, в том числе, и за счет удлиненный цепочки шлюзования вызова через JNI. Однако более или менее существенные временные затраты уходят только на вызов/возврат библиотечного метода, причем телом метода является нативный код, а это значит, что вынесение критичного для производительности кода в нативные библиотеки практически не скажется на производительности, а возможности почти идеальной переносимости сохранятся.

Заметим, что в данной публикации не представлены результаты вспомогательных оценок переключения процессов, что вполне возможно в многозадачной среде, хотя соответствующее исследование проведено. Подлежат дальнейшему изучению вопросы, связанные с получением средних и наихудших вероятностных оценок влияния работы диспетчера задач на точность измерения времени.

Исследование проведено с использованием высокопроизводительной кластерной системы HP BladeSystem Class C (C7000) на основе лезвий C460x2 Intel Xeon 2,33ГГц QuadCore. на кафедре Математического обеспечения и применения ЭВМ Технологического института ЮФУ. В качестве операционных систем использовались Windows XP SP2, Linux OpenSuse 11, SLES 10SP2. Средства разработки gcc (SUSE Linux) 4.3.1 20080507 (prerelease) [gcc-4_3-branch revision 135036], Java™ SE Runtime Environment (build 1.6.0_12-b04), Microsoft Visual Studio 2005, Version 8.0.50727.42 (RTM.050727-4200).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *J.M. Lambert, J.F.* Power. Platform Independent Timing of Java Virtual Machine Bytecode Instructions, Electron. Notes Theor. Comput. Sci. 220 (2008). – С. 97-113.
2. *D.J. Lilja.* Measuring Computer Performance: A practitioner's guide // Cambridge University Press. – 2000.

Хашковский Валерий Валерьевич
Технологический институт федерального государственного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге.
E-mail: vvx@soft.ru.
347939, г. Таганрог, ул. Чехова 346, кв. 210.
Тел.: 88634371673.
Кафедра математического обеспечения и применения ЭВМ; доцент.

Khashkovsky Valeriy Valerievich
Taganrog Institute of Technology – Federal State-Owned Educational Establishment of Higher Vocational Education "Southern Federal University".
E-mail: vvx@soft.ru.
346, Chekova, Taganrog, 347939, Russia.
Phone: 88634371673.
Department of Computer Software; associate professor.

Лутай Владимир Николаевич
Технологический институт федерального государственного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге.
E-mail: vlutay@mail.ru.
347922, г. Таганрог, ул. Чехова 49, кв. 18.
Тел.: 88634361163.
Кафедра математического обеспечения и применения ЭВМ; доцент.

Lutay Vladimir Nicolaevich
Taganrog Institute of Technology – Federal State-Owned Educational Establishment of Higher Vocational Education "Southern Federal University"
E-mail: vlutay@mail.ru
49, Chekova, Taganrog, 347922, Russia.
Phone: 88634361163.
Department of Computer Software; associate professor.

Юрченко Василий Васильевич
Технологический институт федерального государственного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге.
E-mail: esqayr@inbox.ru.
347939, г. Таганрог, ул. Петровская 9, кв.81.
Тел.: 88634360484
Кафедра математического обеспечения и применения ЭВМ; студент.

Yurchenko Vasily Vasilievich
Taganrog Institute of Technology – Federal State-Owned Educational Establishment of Higher Vocational Education "Southern Federal University"
E-mail: esqayr@inbox.ru.
11, Petrovskay, Taganrog, 347900, Russia.
Phone: 88634360484.
Department of Computer Software; student.

УДК 519.7 : 007 + 06

А.В. Белых, С.М. Ковалев, О.В. Ольховик

СОКРАЩЕНИЕ ИЗБЫТОЧНОТИ СХЕМЫ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ БАЗ ДАННЫХ*

В работе рассматриваются вопросы, связанные с сокращением избыточности схемы объектно-ориентированных баз данных. Представленное решение позволяет помимо сокращения избыточности схемы сократить и избыточность данных, а так же отразить идеи, заложенные при проектировании схемы ООБД, минимальным набором визуальных конструкций.

Объектно-ориентированные базы данных; нормализация; дерево наследования; алгоритм нормализации; сокращение избыточности схемы ООБД.

A.V. Belykh, S.M. Kovalev, O.V. Olhovik

THE REDUCTION OF REDUNDANCY OF THE SCHEME OF OBJECT-ORIENTED DATABASES

In work the questions connected with reduction of redundancy of the scheme of object-oriented databases are considered. The presented decision allows to reduce besides reduction of redundancy of the scheme and redundancy of the data and as to reflect the ideas put at designing of scheme OODB, the minimum set of visual designs.

Object-oriented databases, normalization; inheritance tree; normalization algorithm; reduction of redundancy of the scheme of OODB.

Введение. При проектировании схемы ООБД проектировщик редко может сразу построить схему, которая бы содержала минимальное количество элементов, то есть спроектировать не перегруженную схему. Для устранения избыточности ему требуется пройти не одну итерацию процесса проектирования, и при этом не всегда удается устранить все избыточные элементы в схеме, сохранив при этом общую идею. Однако наличие таких элементов в схеме ООБД зависит не только от проектировщика, но и от того визуального языка, на котором ведется проекти-

* Работа выполнена при поддержке: РФФИ (проекты № 07-01-00075, № 09-07-00192).