МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

Юго-Западный государственный университет

(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«__» _____ 2016 г.

МЕТОДИКА ИЗМЕРЕНИЯ ВРЕМЕНИ ВЫПОЛНЕНИЯ ЗАДАННОГО ФРАГМЕНТА ПРОГРАММЫ

Методические указания по выполнению лабораторных и практических работ для студентов направлений подготовки 09.03.01 и 09.04.01

Курск 2016

УДК 621.3
Составитель: Э.И. Ватутин

Рецензент
Кандидат технических наук, доцент *А.Г. Спеваков*

Методика измерения времени выполнения заданного фрагмента программы: методические указания по выполнению лабораторных и практических работ по дисциплине «Параллельное программирование» / Юго-Зап. гос. ун-т; сост.: Э.И. Ватутин; Курск, 2016. 13 с.

Методические рекомендации содержат сведения по разработке программных средств, направленных на измерения затрат времени выполнения заданных фрагментов анализируемой программы на современных языках программирования высокого уровня.

Предназначены для студентов направлений подготовки 09.03.01 и 09.04.01 «Информатика и вычислительная техника».

Текст печатается в авторской редакции

Подписано в печать _____. Формат 60x84 1/16.
Усл. печ. л. Уч. – изд.л. Тираж 30 экз. Заказ . Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул. 50 лет Октября, 94.

Содержание

Основные теоретические положения

Цель работы: познакомиться с методами профилирования программ путем измерения длительности выполнения заданных фрагментов кода.

Измерение времени выполнения заданного участка кода является первым этапом оптимизации и распараллеливания последовательной программы. Его целью является отыскание т.н. узких мест (англ. hotspot или bottleneck), на выполнение которых тратится максимальное количество времени.

Самым простым способом измерения времени выполнения интересующего фрагмента программы является непосредственное измерение времени, затраченного на выполнение данного участка программы. Для получения текущего времени можно воспользоваться WinAPI-функцией `GetTickCount()` (Windows), возвращающей время с момента старта операционной системы в миллисекундах. В Delphi удобным является использование функции `Now()`, определенной в модуле `SysUtils.pas` и возвращающей значение типа `TDateTime`. Значение типа `TDateTime` может быть преобразовано в значение типа `TTimeStamp` при помощи вызова функции `DateTimeToTimeStamp()`. Основное удобство работы с типом `TTimeStamp` заключается в том, что он представляет собой запись с двумя полями: `Time` – время в миллисекундах с полуночи и `Date` – число дней с начала нашей эры.

Delphi

```
program TimeMeasure;
{$APPTYPE Console}

uses
  SysUtils;

var
  StartTime, EndTime: Integer;

begin
  // Начальное значение времени
  StartTime :=
DateTimeToTimeStamp(Now()).Time;
```

C++

```
#include <windows.h>
#include <iostream>

void main()
{
  // Начальное значение времени
  int StartMs = GetTickCount();

  // Анализируемый фрагмент кода
  ...

  // Конечное значение времени
  int EndMs = GetTickCount();
```

```
  // Анализируемый фрагмент кода
  ...

  // Конечное значение времени
  EndTime := DateTimeToTimeStamp(Now()).Time;

  // Время выполнения
  Writeln('Elapsed time: ',
      EndTime-StartTime, ' ms'
  );
  Readln;
end.
```

```
  // Время выполнения
  std::cout << "Elapsed time: " <<
    EndMs-StartMs << " ms\n";
  std::getchar();
}
```

Основным недостатком использования функций, непосредственно основанных на получении системного времени, является их большая погрешность, составляющая 10–20 мс. Они могут быть использованы только для измерения больших временных интервалов (не менее 1 секунды).

Другим способом измерения времени является использование счетчика TSC (Time Stamp Counter), представляющего собой 64-битный MSR-регистр (Model Specific Register) в составе современных процессоров. В момент включения компьютера его значение обнуляется, по приходу каждого нового импульса от тактового генератора значение счетчика инкрементируется. Большой диапазон значений счетчика гарантирует отсутствие его переполнения за время работы компьютера (для процессора, работающего на частоте 4 ГГц, время, необходимое для переполнения счетчика, составляет приблизительно 136 лет). Счетчик TSC может быть использован для измерения времени выполнения заданного фрагмента программы с высокой точностью (до 50–100 тактов или 10–20 нс). Для чтения значения счетчика существует команда RDTSC, позволяющая получение значения счетчика в паре регистров EDX:EAX. Данная команда не является привилегированной и допускает чтение счетчика из программы, запущенной на третьем кольце защиты (ring 3). Для чтения значений большинства других MSR-регистров необходимо находиться на нулевом кольце защиты (ring 0), что невозможно из обычной программы и требует разработки специального драйвера. Пример кода, позволяющий осуществление измерения времени выполнения фрагмента программы с использованием счетчика TSC, приведен ниже:

Delphi

```
program TimeMeasure;
{$APPTYPE Console}

var
  StartTscValue, EndTscValue: Int64;

begin
  // Начальное значение счетчика
  asm
    RDTSC
    MOV     DWORD PTR [StartTscValue], EAX
    MOV     DWORD PTR [StartTscValue+4], EDX
  end;

  // Анализируемый фрагмент кода
  ...

  // Конечное значение счетчика
  asm
    RDTSC
    MOV     DWORD PTR [EndTscValue], EAX
    MOV     DWORD PTR [EndTscValue+4], EDX
  end;

  // Время выполнения
  Writeln('Elapsed time: ',
      EndTscValue-StartValue, ' clocks'
  );
  Readln;
end.
```

C++

```
#include <iostream>

void main()
{
  __int64 StartTscValue, EndTscValue;

  // Начальное значение счетчика
  __asm
  {
    RDTSC
    MOV   DWORD PTR [StartTscValue], EAX
    MOV   DWORD PTR [StartTscValue+4], EDX
  }

  // Анализируемый фрагмент кода
  ...

  // Конечное значение счетчика
  __asm
  {
    RDTSC
    MOV   DWORD PTR [EndTscValue], EAX
    MOV   DWORD PTR [EndTscValue+4], EDX
  }

  // Время выполнения
  std::cout << "Elapsed time: " <<
    EndTscValue - StartTscValue << "
clocks\n";
  std::getchar();
}
```

Получение значения счетчика TSC удобно оформить в виде функции. В Delphi соглашение о вызове (calling convention) `register`, используемое по умолчанию, для функции с типом результата `Int64` обеспечивает возврат значения в паре регистров EDX:EAX, что не требует добавления дополнительных действий в тело функции. В C++ аналогичным образом можно использовать соглашения о вызове `__cdecl` и `__stdcall`.

Delphi

```
function GetTSC(): Int64; register;
asm
  RDTSC
end;
```

C++

```
__int64 __cdecl GetTSC()
{
  __asm RDTSC
}
```

Число тактов, которое занимает выполнение WinAPI-функции `Sleep(1000)` (задержка выполнения потока приблизительно на 1000 мс), приблизительно соответствует текущей тактовой частоте ядра процессора, на котором производится выполнение.

Еще одним способом измерения времени выполнения фрагмента программы является использование WinAPI-функций `QueryPerformanceCounter()` и `QueryPerformanceFrequency()`. Функция `QueryPerformanceCounter()` получает текущее значение высокоточного счетчика, функция `QueryPerformanceFrequency()` возвращает число отсчетов выбранного счетчика в секунду.

Delphi

```
program TimeMeasure;
{$APPTYPE CONSOLE}

uses
  Windows;

var
  StartValue, EndValue, Freq: Int64;
  s: Double;

begin
  // Начальное значение счетчика
  QueryPerformanceCounter(StartValue);

  // Анализируемый фрагмент кода
  ...

  // Конечное значение счетчика
  QueryPerformanceCounter(EndValue);

  // Получание частоты срабатывания счетчика
  QueryPerformanceFrequency(Freq);

  // Расчет времени в секундах
  s := (EndValue - StartValue) / Freq;

  Writeln(EndValue - StartValue, ' ticks');
  Writeln(s:10:10, ' s');

  Readln;
end.
```

C++

```
include <windows.h>
#include <iostream>

void main()
{
  // Начальное значение счетчика
  __int64 StartTicks;
  QueryPerformanceCounter(
    (LARGE_INTEGER *)&StartTicks
  );

  // Анализируемый фрагмент кода
  ...

  // Конечное значение счетчика
  __int64 EndTicks;
  QueryPerformanceCounter(
    (LARGE_INTEGER *)&EndTicks
  );

  // Получение частоты срабатывания счетчика
  __int64 Freq;
  QueryPerformanceFrequency(
    (LARGE_INTEGER *)&Freq
  );

  // Расчет времени в секундах
  double s = (EndTicks - StartTicks) / Freq;

  std::cout << "Elapsed time: \n" <<
    EndTicks-StartTicks << " ticks\n" <<
    s << " s\n";
  std::getchar();
}
```

При выполнении программы на некоторых многоядерных процессорах возможна ситуация, когда значения счетчика TSC для разных ядер различаются. В этом случае если поток начал выполнение на одном ядре в момент времени t_1, а завершил на другом в момент времени t_2, значение времени выполнения $\Delta t = t_2 - t_1$ может быть некорректным (например, отрицательным). Чтобы измерить время выполнения фрагмента в данном случае, необходимо осуществить жесткую привязку потока к одному из ядер путем вызова WinAPI-функции `SetThreadAffinityMask()`:

```
SetThreadAffinityMask(GetCurrentThread(), 1);
```

Первый параметр функции определяет дескриптор потока, для которого осуществляется привязка (в данном примере, текущий поток); второй – битовую маску ядер, на которых разрешено выполнение потока (например, $1_{10} = 0001_2$ – привязка к первому ядру, $2_{10} = 0010_2$ – ко второму ядру, $5_{10} = 0101_2$ – к первому и третьему ядрам).

Если во время выполнения профилируемого потока управление передается на другой поток (например, возникло прерывание или был запущен другой более приоритетный поток), то величина суммарного времени выполнения окажется завышенной, т.к. будет включать время выполнения других потоков (погрешность), не имеющих отношения к профилируемому потоку. Для минимизации влияния данной ситуации необходимо либо произвести несколько замеров (если это возможно) с обработкой полученных результатов (например, расчетом среднего арифметического от полученных значений или выбором минимального), либо вручную повысить приоритет профилируемого процесса путем вызова WinAPI-функции `SetPriorityClass()`:

```
SetPriorityClass(GetCurrentProcess(), REALTIME_PRIORITY_CLASS);
```

Первый параметр функции определяет дескриптор процесса, второй – приоритет. Рекомендуемым приоритетом является `HIGH_PRIORITY_CLASS`, превышающим приоритет большинства процессов в системе. В исключительных случаях возможно использование параметра `REALTIME_PRIORITY_CLASS`, однако в данном случае некоторые системные процессы могут не получать достаточного количества квантов процессорного времени, что может приводить, например, к невозможности обработки системой событий от мышки или клавиатуры. В

случае, если профилируемой фрагмент программы «повиснет», систему придется перезагружать с использованием кнопки Reset на системном блоке.

При необходимости можно дополнительно поднять приоритет потока, выполняемого в рамках процесса, путем вызова WinAPI-функции `SetThreadPriority()`, однако в большинстве случаев в этом нет необходимости.

Значения времени выполнения одного и того же фрагмента кода, получаемые на различных процессорах, могут отличаться из-за различной микроархитектуры, тактовой частоты, объема кэш-памяти процессора, а также из-за работы функций энергосбережения.

Задание. В соответствии с индивидуальным вариантом произвести измерение времени выполнения заданного фрагмента программного кода с использованием перечисленных выше способов. Каждый замер повторить не менее 5 раз, убедиться в наличии погрешности измерения. Замеры произвести не менее чем на 3 различных процессорах, убедиться в различии получаемых величин.

Пример 1. Измерение малого интервала времени – сортировка массива из $N = 1000$ элементов методом «пузырька».

Анализируемый фрагмент программы:

```
repeat
  Changed := False;
  for I := 1 to N-1 do
    if Arr[I] > Arr[I+1] then begin
      t := Arr[I];
      Arr[I] := Arr[I+1];
      Arr[I+1] := t;
      Changed := True;
    end;
until not Changed;
```

Результаты измерения при помощи `QueryPerformanceFrequency()`:

Замер	1	2	3	4	5
Intel Core 2 Duo E6300 1,86 ГГц	3,578 мс	3,477 мс	3,715 мс	3,899 мс	6,673 мс
AMD Athlon X2 3800+ 2,0 ГГц	3,230 мс	3,224 мс	3,173 мс	3,293 мс	3,211 мс
Intel Xeon 5530 2,4 ГГц	3,494 мс	3,540 мс	3,526 мс	3,430 мс	3,491 мс

Результаты почти совпадают на различных процессорах из-за того, что скорость выполнения сортировки определяется скоростью подсистемы памяти (в данном случае, кэш-памяти).

Результаты измерения при помощи счетчика TSC:

Замер	1	2	3	4	5
Intel Core 2 Duo E6300 1,86 ГГц	3,246 мс	3,215 мс	3,168 мс	3,217 мс	3,210 мс
AMD Athlon X2 3800+ 2,0 ГГц	4,248 мс	4,908 мс	7,090 мс	3,819 мс	3,459 мс
Intel Xeon 5530 2,4 ГГц	4,472 мс	4,682 мс	4,684 мс	4,577 мс	4,127 мс

Полученные результаты несколько отличаются от цифр предыдущей таблицы.

Результаты измерения при помощи `GetTickCount()`:

Замер	1	2	3	4	5
Intel Core 2 Duo E6300 1,86	0,000	0,000	0,000	0,000	0,000

ГГц	мс	мс	мс	мс	мс
AMD Athlon X2 3800+ 2,0 ГГц	0,000 мс	0,000 мс	0,000 мс	0,000 мс	0,000 мс
Intel Xeon 5530 2,4 ГГц	0,000 мс	0,000 мс	0,000 мс	0,000 мс	0,000 мс

Время выполнения фрагмента слишком мало для выбранной функции измерения времени.

Пример 2. Измерение большого интервала времени – факторизация числа 352 700 091 909 229 843 (делится на 7 726 079).

Анализируемый фрагмент программы:

```
Divisor := -1;
IsPrime := True;
for I := 2 to Round(sqrt(N)) do
  if N mod I = 0 then begin
    IsPrime := False;
    Divisor := I;
    break;
  end;
```

Результаты измерения при помощи `QueryPerformanceFrequency()`:

Замер	1	2	3	4	5
Intel Core 2 Duo E6300 1,86 ГГц	4,850 с	4,840 с	4,803 с	4,816 с	4,980 с
AMD Athlon X2 3800+ 2,0 ГГц	2,846 с	2,867 с	2,867 с	2,843 с	2,913 с
Intel Xeon 5530 2,4 ГГц	4,567 с	4,568 с	4,627 с	4,569 с	4,570 с

Результаты измерения при помощи счетчика TSC:

Замер	1	2	3	4	5

Intel Core 2 Duo E6300 1,86 ГГц	4,677 с	4,737 с	4,784 с	4,775 с	4,664 с
AMD Athlon X2 3800+ 2,0 ГГц	3,127 с	3,077 с	3,106 с	3,061 с	3,066 с
Intel Xeon 5530 2,4 ГГц	5,879 с	5,867 с	5,871 с	5,872 с	5,873 с

Результаты измерения при помощи `GetTickCount()`:

Замер	1	2	3	4	5
Intel Core 2 Duo E6300 1,86 ГГц	4,703 с	4,719 с	4,750 с	4,813 с	4,625 с
AMD Athlon X2 3800+ 2,0 ГГц	3,016 с	3,000 с	3,000 с	3,015 с	3,016 с
Intel Xeon 5530 2,4 ГГц	4,594 с	4,578 с	4,609 с	4,578 с	4,578 с

Время выполнения приведенного фрагмента кода определяется главным образом временем целочисленного деления (команда `IDIV`). Из результатов измерений видно, что оно различно для различных процессоров.

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Описание условия решаемой задачи в соответствии с индивидуальным вариантом.
4. Листинг программы.
5. Результаты измерения времени выполнения фрагмента программы.
6. Выводы.

Контрольные вопросы

1. С какой целью производится анализ времени выполнения заданных фрагментов программы?
2. Какие способы измерения времени выполнения существуют? В чем их отличия?
3. Для чего применяется изменение приоритета текущего потока или процесса?
4. Для чего применяется привязка потока к одному из ядер процессора?
5. Какие причины приводят к наличию погрешности в измеряемых величинах?

Библиографический список

1. Емельянов С.Г., Ватутин Э.И., Панищев В.С., Титов В.С. Процедурно-модульное программирование на Delphi: учебное пособие. М.: Аргамак-Медиа, 2014. 352 с.
2. Зотов И.В., Ватутин Э.И., Борзов Д.Б. Процедурно-ориентированное программирование на C++: учебное пособие. Курск: КурскГТУ, 2008. 211 с.