提高智能设备应用系统性能的技术研究与设计①

何伟文

(广东工贸职业技术学院, 广州 510510)
(广州城建职业学院, 广州 510925)

摘 要: 随着智能设备在各个领域的广泛应用, 他们的应用系统在功耗、效率、实时性等方面的问题日益突出, 这些问题已经成为智能设备可以持续发展的主要障碍. 如何能够提高智能设备应用系统的性能, 已成为这一领域的一个重要课题. 从应用软件的核心算法、设计架构等方面研究了能够有效提高智能设备应用系统性能的技术. 首先, 基于 JNI、NDK 等技术, 对应用软件的核心算法、架构进行跨平台的移植; 然后使用本地代码(C 或 C++语言)对原来的算法进行改进; 最后, 利用智能设备多核的运算处理能力, 适当地增加任务数量, 改善负载均衡. 实验结果表明: 如果把这些技术适当地集成到原来的应用系统里面, 就能够大幅提高智能设备的整体性能.

关键词: 跨平台移植; 并行计算; 负载均衡; 本地代码

Research and Design on Technology to Improve Intelligent Device Application System Performance

HE Wei-Wen

(Guangdong Polytechnic of Industry & Commerce, Guangzhou 510510, China)
(Guangzhou City Construction College, Guangzhou 510925, China)

Abstract: With the wide application of intelligent devices in all the fields, their application system has the increasingly prominent issues in the aspects such as power consumption, efficiency and instantaneity, and such issues have become the main obstacle for the continuous development of the intelligent devices. How to improve the intelligent device application system performance has become an important topic in this field. In this paper, the technology is able to effectively improve the intelligent device application system performance in the way of core algorithm and design framework of the application software. First, it conducts the cross-platform transplant on the core algorithm and framework of the application software based on the technologies such as JNI and NDK; then, it improves the original algorithm with the native code (C or C++ language); finally, it utilizes the multi-core calculation and processing ability of the intelligent devices to appropriately increase the task quantity and improve the load balancing. The experimental result indicates that: if such technologies are appropriately integrated into the original application system, the overall performance of the intelligent devices can be improved substantially.

Key words: cross-platform transplant; parallel computing; load balancing; native code

1 引言

如今社会, 智能设备的应用范围很广, 像智能家居, 智能穿戴, 智能交通, 智能娱乐等等都是智能设备应用的分类. 然而, 智能设备应用系统是一个资源受限的系统, 他们受限于运算能力、存储空间以及电池续航等多种因数. 其中, 对软件系统的运行空间和时间要求比桌面系统更为苛刻, 因为在智能设备应用的行业内, 竞争空前激烈的情形下, 用户体验作为很容易感受到的主观因素, 越来越受到重视. 比如, 在应用程序的运行过程中, 有时会出现应用没响应的情景, 这就是所谓的 ANR(Application Not Responding), 这样, 用户可以选择“等待”, 让程序继续运行, 或者“强制关闭”该应用了. 一般情况下, 在 android 系统中 Activity 的最长执行时间是 5 秒, BroadcastReceiver 的最长执行时间则是 10 秒. 因此, 针对软件系统的优化, 对智能设备应用系统性能的提高尤显更加重要.

① 收稿时间:2016-01-04;收到修改稿时间:2016-02-25 [doi:10.15888/j.cnki.csa.005324]

然而, 当前能够系统地对这一领域的国内外研究是较为缺失的. 为此, 本文从智能设备的应用软件的核心算法、设计架构等方面研究了能够提高智能设备应用系统性能的技术.

本文通过一个计算圆周率 π 的工程实例, 研究基于应用软件的核心算法、设计架构等方面技术, 分别采用三种设计、实现方式, 阐明这些技术如何能够提高智能设备应用系统的性能.

为了计算圆周率 π, 我们利用如下数学公式:

$$\int_0^1 \frac{1}{x^2+1}dx = \arctan(1) - \arctan(0) = \frac{\pi}{4}$$

将积分公式表示为极限:

$$\pi = 4\int_0^1 \frac{1}{x^2+1}dx = 4\lim_{\Delta x\to 0}\sum \frac{1}{x^2+1}\Delta x$$

实际上 Δx 不可能做到无穷小, 只能让 Δx 尽可能小, 这样求出的结果越接近 π. 我们用 step 表示一个 Δx, 则 num_steps=1/step 尽量地大. 考虑到 f(x) = 1/(x2+1)是一个凸函数,这里取一个中值来求和,即使用 f[(i+0.5)/(num_steps)]来代替 f[i/(num_steps)]求和, 这样求出的和不会总是比实际值偏小. 最后得出设计软件程序所依据的公式(1). 根据这个公式我们设计出本研究所使用的实验平台以及计算程序.

$$\frac{\pi}{4} \approx \sum_{i=0}^{num\_steps} f(x)\times step = \sum_{i=0}^{num\_steps} f(\frac{i+0.5}{num\_steps})\times step = step \times \sum_{i=0}^{num\_steps} f(i+0.5)\times step \quad (1)$$

2　实验平台的创建—新建一个计算圆周率π值的移动应用软件APP

2.1 创建应用软件 APP 的主界面.

应用软件名称为 NdkExp, 我们的实验平台选择一台华为公司的双核智能手机(智能设备), 其中程序的主界面如图 1 所示.

图 1　应用软件的主界面

该程序的主界面中包括“开始 Java 任务”“开始 C 任务”“开始另一种 C 任务”“退出应用”四个按钮.

① 点击“开始Java任务”按钮后, 将开始一个使用纯 Java 语言编写的计算圆周率 π 的任务(简称 Java 任务). 该任务完成后将在该按钮下面显示计算结果和所花的时间(12.52 毫秒), 如图 2 示.

② 点击“开始C任务”按钮后, 将开始一个用C语言编写的方法计算圆周率π的任务(简称C任务). 该任务完成后将在该按钮下面显示计算结果和所花的时间(7.32 毫秒), 如图 2 所示.

③ 点击“开始另一种C任务”按钮后, 将开始一个用 C 语言编写、结合多线程技术计算圆周率 π 的任务(简称: 多线程C任务). 该任务完成后将在该按钮下面显示计算结果和所花的时间(5.63 毫秒), 如图 2 所示.

④ 点击“退出应用”按钮将退出应用.

图 2　三种设计技术的实验结果

2.2 编写应用软件的主程序

应用软件的主程序文件 MainActivity.java, 让其控制整个程序的运行, 并显示计算结果. 主要代码如下:

```
package com.example.ndkexp;
import android.os.Handler;
import android.os.Message;
```

```
……
public class MainActivity extends Activity {
private JavaTaskThread javaTaskThread = null;
private CCodeTaskThread cCodeTaskThread = null;
private AnotherCCodeTaskThread anotherCCodeTaskThread = null;
    private long end_time;
    private long time;
    private long start_time;
……
    private void startJavaTask() {
    //①启动纯 java 代码计算 PI 值.
    if (javaTaskThread == null)
    javaTaskThread = new JavaTaskThread(mHandler);
    if (! javaTaskThread.isAlive())    {
start_time = System.currentTimeMillis();
        javaTaskThread.start();
    tv_JavaTaskOuputInfo.setText("Java 任务正在运行中...");
    }
    }
    private void startCCodeTask() {
    //②启动 C 代码计算 PI 值.
    if (cCodeTaskThread == null)
        cCodeTaskThread = new CCodeTaskThread(mHandler);
    if (! cCodeTaskThread.isAlive())
    {
    start_time= System.currentTimeMillis();
        cCodeTaskThread.start();
tv_CCodeTaskOuputInfo.setText("C 代码任务正在运行中...");
    }
    }
    private void startThirdPartyTask()
    { //③启动多线程 C 代码计算 PI 值.
    if (anotherCCodeTaskThread == null)
    anotherCCodeTaskThread = new AnotherCCodeTaskThread(mHandler);
   if(! anotherCCodeTaskThread.isAlive())    {
       start_time= System.currentTimeMillis();
anotherCCodeTaskThread.start();
            tv_AnotherCCodeTaskOuputInfo.setText("第三方任务正在运行中...");
    }
    }
    ……
```

在启动任务线程的同时,将机器的当前时间记录在 start_time 中; 而当收到任务线程运行完毕的消息后,将机器的当前 时间记录在 end_time 中,两者相减得到任务的运行时长 time.

3 利用实验平台对三种软件设计技术进行分析

3.1 使用纯 Java 语言程序计算圆周率 π

在应用程序 NdkExp 中新建用于计算圆周率 π 的线程任务类 javaTaskThread. 主要代码如下:

```
1  package com.example.ndkexp;
2  import android.os.Handler;
3  import android.os.Message;
4
5  public class JavaTaskThread extends Thread {
6      private Handler mainHandler;
7      public static final int MSG_FINISHED = 1;
8      private static final long num_steps = 100000000;
9      private static final double step = 1.0 / num_steps;
10     private static double pi = 0.0 ;
11
12
13     static String msTimeToDatetime(long msnum){
14         long hh,mm,ss,ms, tt= msnum;
15         ms = tt % 1000; tt = tt / 1000;
16         ss = tt % 60; tt = tt / 60;
17         mm = tt % 60; tt = tt / 60;
18         hh = tt % 60;
19
20         String s = "" + hh +"小时 "+mm+"分 "+ss + "秒 " + ms +"毫秒";
21         return s;
22     }
23
24
25     public void run()
26     {
27         double x, sum = 0.0;
28         long i;
29
30         for (i=0; i< num_steps; i++){
31             x = (i+0.5)*step;
32             sum = sum + 4.0/(1.0 + x*x);
33         }
34         pi = step * sum;
35
36         Message msg = new Message();
37         msg.what = MSG_FINISHED;
38         Double dPi = Double.valueOf(pi);
39         msg.obj = dPi;
40         mainHandler.sendMessage(msg);
41     }
42
```

第 27 行至第 34 行是根据计算公式书写的计算 π 的代码. 这里 x 变量是函数 f(x)= 1/(x2+1)的自变量 x, sum 是 Σ 的累积变量. 累积完 Σ 后,最后在第 34 行,让 π=step×∑算出最后结果. 【在线程的 run 函数中,一旦计算完成,在第 36 行开始,就向主线程发送计算完成的消息】.程序运行的结果如图 2 所示.

从测试的结果看到, 如果使用纯 Java 语言编写的程序, 计算圆周率 π 所花费的时间是: 12.52 秒.

3.2 使用本地代码(C 或 C++语言)计算圆周率 π

这里, 我们基于 JNI、NDK 等技术, 对智能设备应用软件的核心算法、设计架构进行跨平台的移植, 就

是使用 C 或 C++语言对原来的程序算法、设计架构进行改进, 并集成到原来的应用系统里面.

3.2.1 JNI(Java Native Interface)技术

JNI 是 Java 的本地接口, 目的在于利用本地代码, 为 Java 程序提供更高效, 更灵活的拓展[1]. 应用场景包括: 对运行效率敏感的算法实现、跨平台应用移植、调用系统的底层驱动、调用硬件等.

Java 的跨平台, 是以牺牲效率换来对多种平台的兼容性, 因而 JNI 可以说是 Java 短板的补充.

JNI 是 Java Code 和 C/C++ Code 联系的桥梁(如图 3).

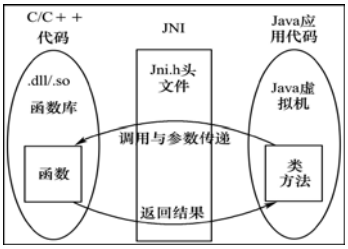


图 3　JNI 工作流程图

3.2.2 NDK(Native Development Kit) 技术

NDK 提供了一系列的工具, 开发者可以开发 C(或 C++)的动态库, 并能将 so 和 java 应用一起打包成 apk[2].

NDK 开发的流程框架如图 4 所示. 我们以 Android 系统为例, 一个 Android 应用由 Android 应用文件、Java 本地库文件和动态库等三部分组成. 这三部分是由不同的来源经过各自的生成路径得来的. 其中 Android 应用文件和 Java 本地库文件由 Android_SDK 生成. 而动态库文件, 以.so 为文件后缀名,由非本地运行代码(典型的是 C 源代码文件),通过 NDK 生成. 最后三者安装到目标机上, 协同完成应用的运行.

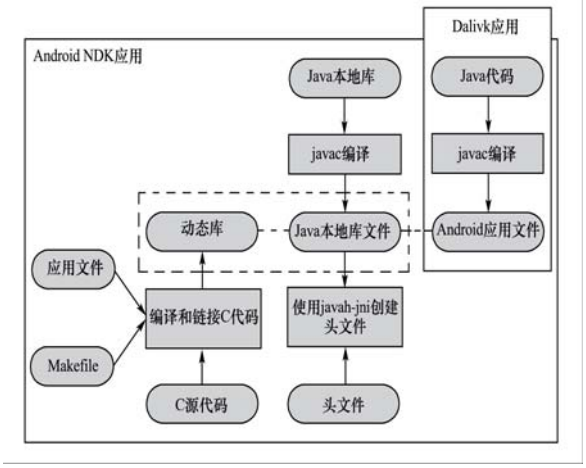


图 4　NDK 开发的流程框架

3.2.3 基于 JNI、NDK 技术, 提高智能设备软件系统的性能

下面, 我们通过拟建的实验模型来分析研究基于 JNI、NDK 技术, 如何能够提高智能设备软件系统的性能.

(1) 在工程中新建调用本地函数来计算圆周率 π 的线程任务类 CCodeTaskThread, 其源代码内容如下:

```
1  package com.example.ndkexp;
2  import android.os.Handler;
3  import android.os.Message;
4  public class CCodeTaskThread extends Thread {
5      private Handler mainHandler;
6      public static final int MSG_FINISHED = 2;
7      private native double cCodeTask();      // 调用外部的C函数来完成计算任务
8      static String msTimeToDatetime(long msnum)
17     @Override
18     public void run()
19     {
20         double pi = cCodeTask();            // 调用外部的C函数来完成计算
21         Message msg = new Message();
22         msg.what = MSG_FINISHED;
23         Double dPi = Double.valueOf(pi);
24         msg.obj = dPi;
25         mainHandler.sendMessage(msg);
26     }
```

以上代码与使用纯 Java 语言编写的线程任务类 javaTaskThread 的代码框架有所不同, 我们将第 20 行, 原本计算圆周率 π 的 Java 代码替换为调用本地函数 cCodeTask 来实现.

(2) 用本地代码(C 语言)编写 cCodeTask 函数

将 cCodeTask 函数编译成 so 库文件,主要的设计步骤如下:

① 建立 C 接口文件.

由于本例是 CCodeTaskThread 类使用本地函数,因此要根据该类的 class 文件来生成头文件. 在命令行中进入工程的目录,然后运行如下命令:

E:\temp\Android Dev\workspace\NdkExp> javah -classpath "D:\Android\android-sdk\platforms\android-15\android.jar";bin/classes com.example.ndkexp.CCodeTaskThread

该命令会在工程目录下生成一个名为: com_example_ndkexp_CCodeTaskThread.h 的文件, 该文件的主要内容为:

……

27　JNIEXPORT jdouble JNICALLJava_com_example_ndkexp_CCodeTaskThread_cCodeTask

28　(JNIEnv* ,jobject);

……

第 27、28 行,定义了本地函数 cCodeTask 的原型.

② 根据上述头文件,在工程的 jni 目录中建立相应的 C 代码文件, 本例取名为 mycomputetask.c, 其内容如下:

```
1  #include <jni.h>
2  jdouble  Java_com_example_ndkexp_ CCodeTask Thread_cCodeTask (JNIEnv*env, jobject thiz)
3  {
4  const long num_steps =100000000;
5  const double step = 1.0/num_steps;
6  double x,sum = 0.0;
7  long i;
8  double pi = 0;
9  for(i=0;i< num_steps;i++){
10  x = (i+0.5)*step;
11  sum = sum + 4.0/(1.0 + x*x); }
12  pi = step* sum;
13  return(pi); }
```

上述代码的第 4 行到第 13 行是函数体, 用于计算 π 的数值.

③ 在工程的 jni 目录下,建立 Android.mk 和 Application.mk 文件. 其中 Android.mk 的内容如下:

```
1  LOCAL_PATH := $(callmy-dir)
2  include $(CLEAR_VARS)
3  LOCAL_MODULE := ndkexp_extern_lib
4  LOCAL_SRC_FILES := mycomputetask.c
5  include $(BUILD_SHARED_LIBRARY)
```

其中第 4 行指明本例的 C 代码文件, 第 3 行指明生成的库文件名, 该名必须与工程的 MainActivity.java 文件的 System.loadLibrary 函数的参数一致.

④将 C 代码文件编译成工程 lib 目录下的 so 库文件.

应用运行的界面如图 2 所示. 从测试的结果看到, 如果使用本地代码编写的程序, 计算圆周率 π 所花费的时间是: 7.32 秒.

3.3 使用本地代码(C 或 C++语言)及多线程技术计算圆周率 π

1. 设计思路

由于智能设备的多核处理器支持多线程在物理上的并行运行, 例如, 对于上例的运行环境, 这个多核处理器就可以支持两个线程的并行运行. 这就是我们算法改进的切入点: 我们采用“分治法”的思路, 设法让计算任务分摊到多个(本例是两个)线程中来运行, 这样并行计算的线程就可以加快运行速度.

2. 实现的步骤如下:

首先, 修改在工程的 jni 目录中建立相应的 C 代码文件, 本例取名为 mycomputetask_2.c, 主要代码如下:

```
……
    const long num_steps = 100000000;
    const int gNumThreads = 2;
    double gStep = 1.0/ num_steps
    double gPi = 0.0;
    CRITICAL_SECTION gCS;
DWORD WINAPI threadFunction(LPVOID   pArg){
     int myNum = *((int *)pArg);
     double Sum = 0.0, x;
    for ( int i=myNum; i<num_steps; i+=gNumThreads )
//每次有 gNumThreads 个线程 运行
     {    x = (i + 0.5f) / num_steps;
          Sum += 4.0f / (1.0f + x*x);   //在每个线程计算 PI 值  }
     EnterCriticalSection(&gCS);
     gPi += Sum * gStep;                  //将每个线程的计算结果累加到最后结果 gPi
     LeaveCriticalSection(&gCS);
     return (gPi); }
```

然后, 按照 3.2.3 中(2)的方法, 将 C 代码文件编译为工程 lib 目录下的 so 库文件. 应用运行的界面如图 2 所示. 从测试的结果看到, 如果使用本地代码(C 语言)及多线程技术编写的程序, 计算圆周率 π 所花费的时间是: 5.63 秒.

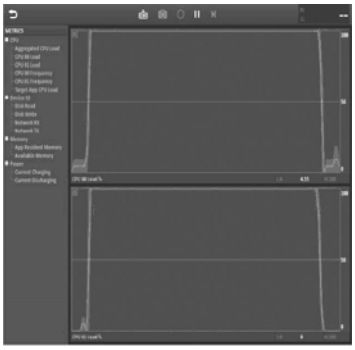
3. 使用 GPA(Graphics Performnce Analyzers)工具来验证与评估应用软件的性能变化

下面使用 GPA 来分析此应用软件, 分析的结果如图 5 所示.

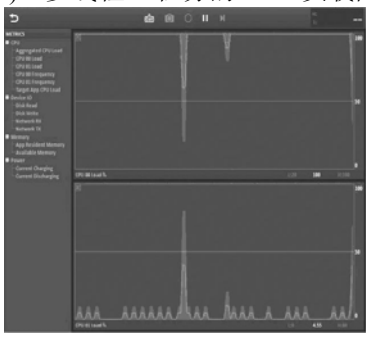
从图 5(a)中可以看出,点击“开始另一种 C 任务”按钮,计算线程运行开始后,两个 CPU 的负载都从低负载立即上升到满负荷状态. 计算完成后,两个 CPU 的负载都重新回到低负荷状态. 在计算任务运行期间,全部的 CPU 都处于满负荷状态, 不存在负载在两个 CPU 之间轮流的情况.

而在图 5(b)的单线程 C 任务, 计算线程运行时,两个 CPU 中始终只有一个 CPU 是满负荷的,另一个处于

低负荷水平, 也就是说总的负载率只有 50%. 这就是改进应用软件的核心算法、程序设计架构之后, 系统的运行速度提高的内在原因.



(a)　多线程 C 任务的 CPU 负载图



(b)　单线程 C 任务的 CPU 负载图

图 5 GPA 的分析结果

4　三种软件设计技术的性能评价

本文结合一个工程实例---计算圆周率 π 的值, 分别采用三种软件设计、实现技术: ①用纯 Java 语言编写; ②用C语言编写; ③用C语言与多线程技术, 来完成相同的任务. 他们所耗费的时间如图 6 所示.

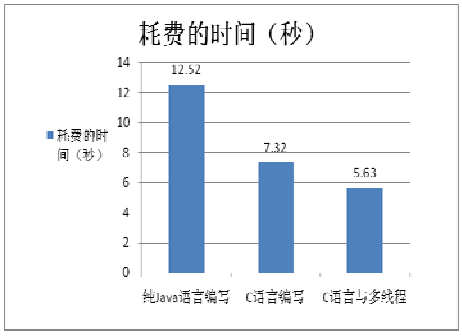


图 6　三种软件设计技术对比图

从图 6 可以看到: 第②种方法比第①种效率提升了 41.44%; 第③种方法比第①种提升了 55.03%; 第③种方法比第②种又提升了 23.09%.

从上述的实验数据、图表, 我们可以得出以下结论: 如果能够把 NDK、本地代码与多线程等技术适当地运用到原来的智能设备应用系统里面, 就能够大幅减少系统的响应时间, 从而有效提高他们的整体系统性能.

5　结语

本文主要研究了智能设备应用软件的核心算法、设计架构等方面能够提高智能设备应用系统性能的技术. 随着智能设备在各个领域的广泛应用, 由于智能设备应用系统存在功耗、效率、实时性等要求,这就需要该系统能够对任务进行有效的管理,如分配调度、通信、同步等,以实现多核处理器的最大化利用. 因此, 需要具有多处理器的芯片通过并行计算方式组织为多核系统平台, 实现智能设备高性能处理能力. 与此同时, 以本地代码的形式实现应用系统中对性能要求较高的运算, 也可以提高智能设备的处理能力. 所以正确使用本地代码, 就能在智能设备中构建出高性能的应用软件, 如硬件视频编码和解码、图形处理(如: 游戏开发)和大数据密集型计算等等.

本文通过这些关键技术的研究工作及成果,可以为智能设备系统的研究与开发提供一些帮助,以促进智能设备的应用推广, 具有一定的参考价值.

参考文献

1 ttp://www.cnblogs.com/devinzhang/archive/2012/02/29/237372 9.html.

2 http://developer.android.com/sdk/ndk/index.html.

3 Rakvic Q, Cai J. González G, Magklis P, Chaparro A. González. Thread-management techniques to maximize efficiency in multicore and simultaneous multithreaded microprocessors. ACM Trans. on Architecture and Code Optimization (TACO), 2010, (2).

4 王晶,樊晓桠,张盛兵,等.多核多线程结构线程调度策略研究.计算机科学,2007,34(9):256-258.

5 李涛,高德远,樊晓娅,等.高性能微处理器性能模型设计.航空电子技术,2011,2:25–28.

6 ShahidBokhari JS. Exploring the performance of massively multithreaded architectures. Concurrency Computation: Pract. Exper, 2010, (5).

7 Eyerman S, Eeckhout L. Modeling critical sections in

Amdahl’s law and its implications for multicore design. ACM SIGARCH Computer Architecture News, 2010, (3).

8 戴鸿君.基于异构多核体系与组件化软件的嵌入式系统研究[博士学位论文].杭州:浙江大学,2007.

9 黄永兵,陈明宇.移动设备应用程序的体系结构特征分析.计算机学报,2015,38(2):386–396.

10 王丰.基于云计算的账户管理系统及移动应用的研究与实现[硕士学位论文].北京:北京邮电大学,2015.

11 毛玉越,路莹,刘娜,等.Android 技术在大型设备智能维护系统中的应用.大连工业大学学报,2014(6).