(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 103473168 A
(43)申请公布日 2013.12.25

(21)申请号 201310415538.4

(22)申请日 2013.09.12

(71)申请人 中国科学院声学研究所
地址 100190 北京市海淀区北四环西路 21 号

(72)发明人 朱浩 彭楚 应欢 王东辉 洪缨

(74)专利代理机构 北京亿腾知识产权代理事务所 11309
代理人 陈霁

(51)Int.Cl.
G06F 11/34(2006.01)

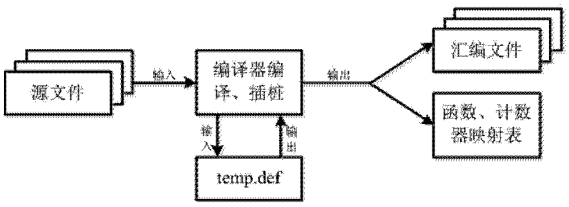
权利要求书1页 说明书4页 附图2页

(54)发明名称

一种热点程序的统计方法

(57)摘要

本发明公开了一种热点程序的统计方法，该方法步骤包括：对源文件进行伪指令插桩，所述伪指令作为源文件中各函数的第一条指令，所述伪指令为第一伪指令，所述第一伪指令的操作数代表了各函数对应计数器在数据段中的偏移量，所述数据段用于记录各函数被访问次数和/或各函数被执行的时间；所述源文件经过编译后输出携带所述第一伪指令的目标文件，当执行所述第一伪指令时，根据所述第一伪指令中的操作数查找数据段中对应的全局计算器并加1，而在目标文件的返回指令中添加对各函数执行时间的记录；当目标文件运行完成时，根据数据段中存储的数据获取各函数被运行的次数。本发明实现简单、便于移植。



CN 103473168 A

1. 一种热点程序的统计方法，其特征在于：对源文件进行伪指令插桩，所述伪指令作为源文件中各函数的第一条指令，所述伪指令为第一伪指令，所述第一伪指令的操作数代表了各函数对应计数器在数据段中的偏移量，所述数据段用于记录各函数被访问次数和/或各函数被执行的时间；

所述源文件经过编译后输出携带所述第一伪指令的目标文件，当执行所述第一伪指令时，根据所述第一伪指令中的操作数查找数据段中对应的全局计算器并加1，而在目标文件的返回指令中添加对各函数执行时间的记录；

当目标文件运行完成时，根据数据段中存储的数据获取各函数被运行的次数。

2. 根据权利要求1所述的方法，其特征在于：从所述数据段选定一个有效的起始地址并划分足够大的内存空间，用于存储所述各函数对应的计数器。

3. 根据权利要求1所述的方法，其特征在于：在对源文件进行编译前读入一个用于存储当前偏移量的临时文件，从中读出当前偏移量，所述偏移量为所述各函数对应计数器与起始地址的距离，所述临时文件在所述源文件编译完毕后，重新被写入新偏移量。

4. 根据权利要求1所述的方法，其特征在于：所述数据段中每8个字节对应于一个目标函数，其中，前4字节代表目标函数被访问的次数，后4字节代表函数被执行的时间。

5. 根据权利要求1所述的方法，其特征在于：所述数据段中各全局计数器与源文件名、函数名称建立一对一的映射关系。

6. 根据权利要求1所述的方法，其特征在于：当所述函数中封装了多个操作或多个语句块时，将第二伪指令分别插入到各语句块的入口出，将第三伪指令分别插入到各语句块的出口处，并通过所述第二伪指令获取语句块的运行时间，根据所述语句块的运行时间和对应函数运行总的时间计算目标语句块的占用率，所述语句块的占用率为：$\frac{\text{目标语句块总时间}}{\text{对应函数总时间}}$，所述目标语句块是由程序员根据需要自行划分。

7. 根据权利要求6所述方法，其特征在于：所述目标语句块的运行时间通过以下步骤获取：

创建一个用于保存执行所述第三伪指令时的时间点，以及根据所述第三伪指令的全局索引值创建数据段；

当执行到所述第三伪指令时，进行一次退栈操作，获取所述目标语句块的运行时间，并根据所述数据段的值完成对相应计时器的累加；

当执行程序执行完成时，顺序读取所述数据段保存的目标语句块的运行时间数据。

8. 根据权利要求1至7中任一权利要求所述的方法，其特征在于：该方法在模拟执行平台上运行，所述模拟执行平台是针对各种体系结构处理器的模拟器。

一种热点程序的统计方法

技术领域

[0001] 本发明涉及程序性能分析技术，尤其涉及一种热点程序的统计方法。

背景技术

[0002] 程序运行时并非均衡的对每个函数、数组进行访问，约80%的运行时间花在20%的代码中。嵌入式平台上的存储资源有限，对热点程序分布的统计是为了将访问频度较高的函数、数组尽可能存储与访问速度更快的片上存储。

[0003] 集成开发环境（Integrated Development Environment，IDE）是用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面工具。集成了代码编写功能、分析功能、编译功能、调试功能等一体化的开发软件服务套。

[0004] 现有的集成开发环境，如微软的Visual Studio系列等，它们运行的底层硬件平台均提供了用于程序性能数据反馈的专用指令，并依托于编译器对目标源程序进行代码插桩（Code Instrumentation），然后依托处理器硬件平台的计算完成对热点程序与数据的统计。

[0005] 现有技术的热点程序方法的不足之处在于：

[0006] 1、依赖于处理器硬件平台的支持；

[0007] 2、不支持对语句块执行时间的评估。

发明内容

[0008] 本发明的目的是为了解决上述现有技术存在的不足之处，提供了一种实现简单、便于跨平台移植的热点程序统计方法。

[0009] 为实现上述目的，本发明提供了一种热点程序的统计方法，该方法包括以下步骤：

[0010] 对源文件进行伪指令插桩，所述伪指令作为源文件中各函数的第一条指令，所述伪指令为第一伪指令，所述第一伪指令的操作数代表了各函数对应计数器在数据段中的偏移量，所述数据段用于记录各函数被访问次数和/或各函数被执行的时间；

[0011] 所述源文件经过编译后输出携带所述第一伪指令的目标文件，当执行所述第一伪指令时，根据所述第一伪指令中的操作数查找数据段中对应的全局计算器并加1，而在目标文件的返回指令中添加对各函数执行时间的记录；

[0012] 当目标文件运行完成时，根据数据段中存储的数据获取各函数被运行的次数。

[0013] 本发明实现简单、不依赖于硬件平台、可完成对语句块执行时间的评估。

附图说明

[0014] 图1为本发明实施例定义的数据段.funcnt结构示意图；

[0015] 图2（a）为本发明实施例提供的指令插桩流程图；

[0016] 图2（b）为temp.def、映射表、funcnt.s文件格式；

[0017] 图 3 为本发明实施例提供的统计目标语句块的执行比率的流程图。

具体实施方式

[0018] 下面通过附图和实施例，对本发明的技术方案做进一步的详细描述。

[0019] 本发明实施例是在处理器模型的基础上讨论如何统计热点程序的统计，所研究的内容并不仅局限于统计热点函数，还提供了对代码片段进行时间开销评估的支持。评估函数被访问次数以及总执行时间是用于评估目标函数是否为热点的两个标准。

[0020] 本发明实施例通过将伪指令 Cnt 插入汇编代码片段中来实现函数被访问次数的统计。Cnt 指令的语法格式如下：

[0021] 语法格式：Cnt Imm21（21 位无符号数）

[0022] Cnt 指令所用到的推断寄存器默认为 Pr0，即确定执行。Imm21 为 Operand 的值，代表 21 位的无符号数。处理器内核模型的虚拟内存系统最大为 4GB，因此，为了简化对函数执行次数统计的难度，本发明实施例定义了一段占据足够字节数的数据段 .funcnt，每 8 个字节对应于一个函数，其中，前 4 字节代表目标函数被访问的次数，后 4 字节代表函数执行的时间。尽管如此，程序设计中不能排除不同源文件中存在由 static 变量定义的相同函数名的可能，因此，.funcnt 段中各全局计数器必须与文件名、函数名称建立一对一的映射关系。具体过程如上所示，而 Cnt 指令语法格式中的 Imm21 代表了各函数对应计数器在 .funcnt 段中的偏移量。

[0023] 为了不给编译器造成压力，Cnt 指令并不参于寄存器分配阶段。因此，编译器对 Cnt 指令进行插桩时，只需将 Cnt 指令作为目标函数的第一条指令。然而，通常编译器是顺序的读入、处理各个文件，即下一次编译无法获知上次编译过程中的中间数据，为了保证各函数内 Cnt 指令的立即数是连续的，编译器在每次编译过程中都会读入一个用于存储当前偏移量的临时文件 temp.def，编译之前从中读出当前偏移量，当源文件处理完毕后写入存放新偏移量的 temp.def 文件。编译结束后，编译器不仅输出插桩后的汇编文件，而且还更新了动态分析过程中用于统计数据的函数、计数器映射表，整个操作流程如图 2(a) 所示。为了使 .funcnt 段参与链接期的内存地址分配，本发明实施例添加了用于声明该段的 funcnt.s 汇编文件，并一起进入后续的汇编、链接阶段。过程中使用到的 temp.def、映射表、funcnt.s 文件的格式如图 2(b) 所示，其中 ByteNum 代表 .funcnt 申请的字节，且 ByteNum= 最大索引号 *8bytes。

[0024] 本发明实施例在动态分析过程中定义了一个全局计时器 Cycler，当执行 Cnt 指令时，根据 Cnt 指令中的立即数查找到 .funcnt 段中对应的全局计数器并加 1，即 Cnt 指令的功能描述如下：

[0025] *(unsigned int*)(addr(.funcnt)+imm*8)+=1;

[0026] Cycler=total_cycles;

[0027] 而在对返回指令 Ret 中添加对每个函数执行时间的记录：

[0028] *(unsigned int*)(addr(.funcnt)+imm*8+4)+=total_cycles-Cycler;

[0029] 最终，当目标程序运行完成，根据 ByteNum 的值顺序读出 .funcnt 段中存储的数据，即动态执行过程中各函数运行次数。表 1 是通过本发明实施例对 1024 点快速傅立叶变换 (Fast Fourier Transform, FFT) 程序进行的函数统计结果。

[0030] 表 1FFT 程序统计结果

[0031]

Function（函数名称）	Executed Counts（执行次数）	DSP Cycles（执行时间）
fft	1	378164
complex_add	5120	107520
complex_sub	5120	107520
complex_swp	496	9424
init_ ata	1	8280
__loader	1	119
main	1	378287

[0032] C 语言是一种面向过程的高级编程语言，由它编写的函数中通常封装了即将顺序执行的多个操作或多个语句块，独立评估这些语句块占用目标函数执行时间的比例有利于程序员编写出更高效的代码。然而，编译器很难恰当的将目标函数切割为多个语句块，这主要因为划分的约束很难有统一的标准，几乎无法满足所有人的要求，划分的太细致无法辅助程序员进行算法层上的改进，太粗糙则失去了评估的意义。因此，本发明实施例在进行片段分析时将划分操作交由程序员实现，并提供特定格式的标识语句，这种方法类似于 OpenMP 中要求程序员对可并行部分进行标识来辅助编译器生成并行代码，OpenMP 是用于共享内存并行系统的多线程程序设计的一套指导性的编译处理方案 (Compiler Directive)。为了尽可能的减少编译器修改的工作量，本发明实施例采用内嵌汇编实现，绝大部份的操作在汇编期实现。在 GCC 编译器架构中该种内嵌汇编的格式为：

[0033] __asm__("::sentence");

[0034] 助记符 "::" 作为汇编期识别的线索，sentence 为注释语句，包括：起始句 "Block Name","Name" 由程序员定义且同一函数内部不能重复定义；终止语句 "EndBlock"。起始句与终止句之间为目标高级语句块对应的汇编指令。程序员编程过程中插入这两条语句时，需保证这两条语句分别位于目标语句块的入口和出口处，且必须保证成对出现，代码 (a) 为一个正确示例，代码 (b) 为插入不完整的错误示例，这将导致无法统计，代码 (c) 为非入口或非出口处的错误示例，这同样导致统计过程中的算法出现错误。

[0035]

代码(a)

```
void Func(void)
{
    int i = 0;
__asm__("::Block Sample")
    for (i=0; i < 100; i++)
        ……
__asm__("::EndBlock")
}
```

代码(b)

```
void Func(void)
{
    int i = 0;
__asm__("::Block Sample")
    for (i=0; i < 100; i++)
        ……
}
```

代码(c)

```
void Func(void)
{
    int i = 0;
__asm__("::Block Sample")
    for (i=0; i < 100; i++)
    {
        ……
__asm__("::EndBlock")
    }
}
```

[0036] 本发明实施例在进行汇编文件分析时提供了一个全局计数器，并逐一将内嵌汇编

语句"Block Name"替换为伪指令Block,语句"EndBlock"替换为EndBlock,其中Block指令的Operand为注释语句的全局索引值idx,EndBlock指令不定义Operand的值,他们的推断寄存器Pr均默认为Pr0,而"Name"则逐一添加入独立的数据段存储在最终的可执行ELF文件内,它们的语法格式如下:

[0037] 语法格式:Block Imm21(21位无符号数)

[0038] EndBlock

[0039] 汇编器根据全局索引值idx创建.blkcnt数据段,申请的字节数位idx*8bytes,其中每8个字节的首4字节对应为各Block伪指令指向的计数器,尾4字节对应为当前函数的全局索引值,并建立funcnt.s文件格式的BlockCnt.s文件由于汇编、链接等后续操作。

[0040] 图3为本发明实施例提供的统计目标语句块的执行比率的流程,具体步骤如下:

[0041] 1、处理器模型创建一个用于保存执行到Block指令时的时间点time(i)以及Block指令中Operand的全局栈结构,即blkidx(i);

[0042] 2、执行到EndBlock时,进行一次退栈操作,获取语句块的运行时间time(i),.查询.blkcnt数据段并根据blkidx(i)的值完成对相应计时器的累加;

[0043] 3、当应用程序执行完成时,顺序读取.blkcnt段保存的运行时间数据,而语句块的占用率即为$\frac{\text{目标语句块总时间}}{\text{对应函数总时间}}$

[0044] 4、从虚拟内存中读出函数名、Block的命名,并以如下格式输出,即在fft函数中的basic语句块占用的比率为11.93%。

[0045] fft→basic→11.93%

[0046] 本发明实施例实现简单、不依赖于硬件平台、可完成对语句块执行时间的评估。

[0047] 显而易见,在不偏离本发明的真实精神和范围的前提下,在此描述的本发明可以有许多变化。因此,所有对于本领域技术人员来说显而易见的改变,都应包括在本权利要求书所涵盖的范围之内。本发明所要求保护的范围仅由所述的权利要求书进行限定。

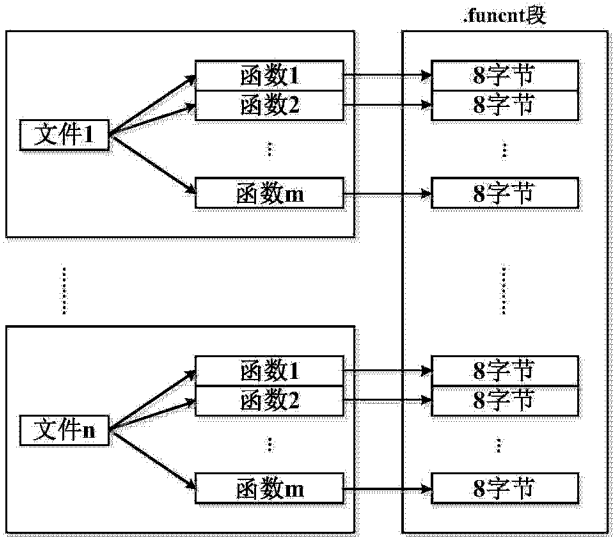

.funcnt段
文件1
函数1
函数2
函数m
8字节
8字节
8字节
文件n
函数1
函数2
函数m
8字节
8字节
8字节

图 1

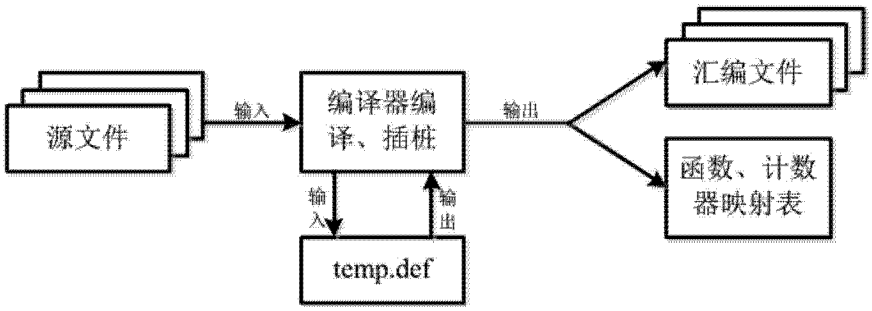

源文件
输入
编译器编译、插桩
输出
汇编文件
函数、计数器映射表
输入
输出
temp.def

图 2（a）

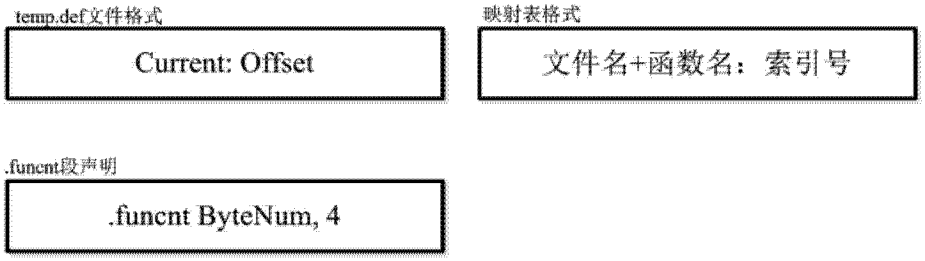

temp.def文件格式
Current: Offset
映射表格式
文件名+函数名：索引号
.funcnt段声明
.funcnt ByteNum, 4

图 2（b）

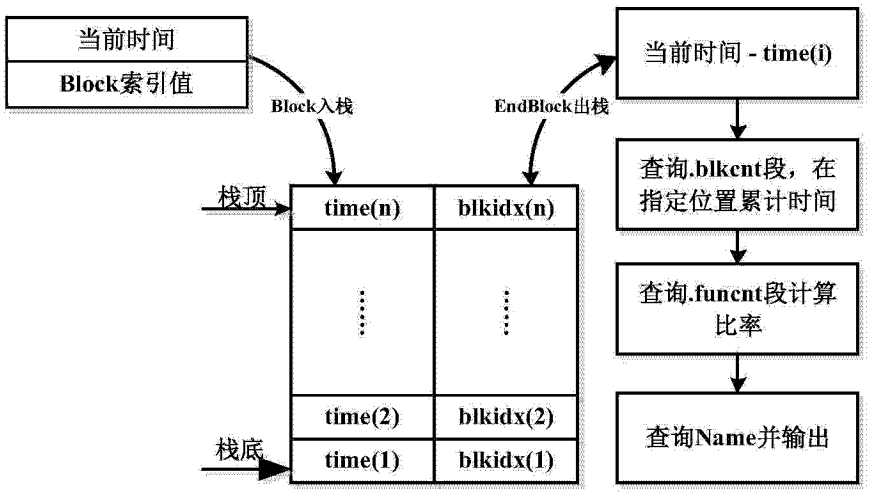

当前时间
Block索引值
Block入栈
EndBlock出栈
当前时间 - time(i)
查询.blkcnt段，在指定位置累计时间
查询.funcnt段计算比率
查询Name并输出
栈顶
栈底
time(n)
blkidx(n)
time(2)
blkidx(2)
time(1)
blkidx(1)

图 3