一种嵌入式系统实时性能分析的融合机制*)

王济勇 林 涛 王金东 韩光洁 赵 海
(东北大学信息科学与工程学院 通信与信息系统研究所 沈阳110004)

摘 要 许多嵌入式系统要求硬实时或者软实时地执行,为了确保这些要求得到满足,不仅要对单个任务或代码段的执行时间进行测量,同时还必须确定整个系统的实时性能。为此,针对导致传统的纯软件方法测量精确度低的两个主要原因,借助于多源数据的信息融合思想,依据改进后的纯软件的实时代码执行时间的测量方法,提出了一种用于嵌入式系统实时性能分析的融合机制,基于这种机制,设计和开发人员能够查明时限错误,找出需要优化的代码以解决期限错过问题,并确定特定环境下实时任务集的可调度性。

关键词 嵌入式系统,纯软件方法,实时性能分析,融合机制

A Fusion Mechanism for Analyzing Real-Time Performance of Embedded Systems

WANG Ji-Yong LIN Tao WANG Jin-Dong HAN Guang-Jie ZHAO Hai
(Institute of Communication & Information System, School of Information Science & Engineering, Northeastern University, Shenyang 110004, China)

Abstract Many embedded systems require hard or soft real-time execution. To ensure the requirements are met, it is necessary to measure the execution time of individual tasks, as well as establish the overall real-time performance of the system. The traditional software-only methods for measuring the execution times of real-time codes are easy to use and low cost, but provide lower resolution and greater overhead than hardware ones, which impedes using it for analyzing real-time performance, such as identifying whether a specific task set is schedulable. In order to overcome the main two obstacles which cause lower accuracy of software-only methods, based on the information fusion idea of multi-source data and using the improved software-only method for measuring the execution times of real-time codes, which also takes the information fusion idea of multi-source data to improve the accuracy of the measurements, while it reserves the good features owned by traditional ones, this paper presents a fusion mechanism for analyzing the real-time performance of embedded systems. Using the mechanism, the designers and developers can pinpoint the timing problems, find the code to be optimized so as to avoid the missed deadlines, and identify the schedulability of a real-time task set under a specific scheduling environment.

Keywords Embedded systems, Software-only method, Real-time performance analysis, Fusion mechanism

1 引言

许多嵌入式系统要求硬实时或者软实时地执行,即必须满足严格的时限约束。实时系统理论提倡使用合适的调度算法,且在建造系统之前完成可调度性分析。单凭这个理论,一个嵌入式系统一般不会正常工作,而且实际的设计和开发人员通常不采用这个理论,因为这样的系统很可能不依据规范工作。理论和实践之间存在一个平衡,用来测量执行时间的系统化的技术和实时系统理论提供指导能够帮助工程人员设计、分析,并快速纠正实时嵌入式系统中的时限错误。

当前的许多测量实时代码执行时间的技术,由于各种原因都存在不同程度的不精确性,在嵌入式系统中,涉及到实时性能分析时,对测量值的精确度有一定的要求,这种不精确性不能忽略。由于测量技术本身的缺陷,单个技术的测量结果一般不能满足对测量值精确度的要求,即使存在这样的技术,在当前情况下,代价也是十分高昂的。因此,在工程实际中,更可取的方法是,采用几种较简单的方法相结合的方式,借助于数据融合的思想[1,2],将几种简单方法的测量结果通过一个融合模型进行相关和组合以达到相应的精确度要求。

一般认为,导致传统的纯软件方法测量精确度低的主要原因有两个,其一是用于收集状态代码的主机实时时钟分辨率较低,其二是测量值中以不一致的方式包含了多种额外开销。为此,本文提出了一种嵌入式系统实时性能分析的融合机制,包括一个纯软件的测量方法和两个数据融合模型。纯软件方法 RTETMeasuring 针对第一个原因采用精确实时时钟测量单个实时任务或者实时代码段的执行时间、实时操作系统(RTOS)开销和状态代码的生成代码开销;融合模型 RTETFusion 从根本上减少 RTETMeasuring 方法测量值因以不一致的方式包含多种额外开销而造成的误差,从而获得满足一定精确度要求的执行时间测量值;基于相应精确度的测量值,融合模型 RTPAFusion 依据相关的数学理论判定相应调度环境下实时任务集的可调度性。因此,通过这种融合机

*)基金项目:国家自然科学基金资助项目(69873007)。王济勇 博士研究生,研究方向:嵌入式 Internet、信息融合。林 涛 博士研究生,研究方向:嵌入式 Internet、信息融合。王金东 博士研究生,研究方向:嵌入式 Internet、信息融合。韩光洁 博士研究生,研究方向:嵌入式 Internet、信息融合。赵 海 博士,教授,博士生导师,研究方向:嵌入式 Internet、信息融合。

制，设计和开发人员能够剖析一个实时系统，过滤和提取测量的数据，分析数据，判定一定调度环境下实时任务集的可调度性和预测对实时系统修改后的结果，据此可以指导他们有目的地优化代码，调整实时任务集和/或扩充硬件资源等，以满足嵌入式系统硬实时或者软实时要求。

2 嵌入式系统实时性能分析的融合机制

本文提出的嵌入式系统实时性能分析的融合机制是以多源数据的信息融合思想为基础，凭借一种代价低且简单的纯软件测量方法从不同的方面测量实时代码的执行时间，其中包括单个任务(或代码段)、实时操作系统任务调度开销、实时操作系统中断调度开销和状态代码的生成代码开销，然后将这些测量值通过计算精确执行时间的融合模型 RTETFusion 的计算，得到满足相应精确度要求的执行时间测量值，进而，以这些数值信息为基础，利用 RTPAFusion 模型对相应的实时代码进行分析并得到指导设计和开发人员调试和调整实时任务的结果。

2.1 执行时间的测量方法

当前存在许多测量实时代码执行时间的方法，就单独一个技术来说，没有最好的。每个技术都在多个测量属性之间做出折衷，这些属性包括，分辨率、精确度、粒度和难度[3]。为了给 RTETFusion 模型提供相应的信息源，本文采用了一种代价低且易于使用的纯软件方法来测量实时用户任务或操作系统代码段的执行时间，然后用一种分析的方法在相应的假设基础上，测量实时操作系统的任务调度开销和实时操作系统中断调度开销以及状态代码的生成代码开销。

1 *纯软件的执行时间测量方法* RTETMeasuring 其特点是成本低、操作简单、方便，基本不需要额外的硬件支持，但缺点是，精确度低。为了解决导致传统的纯软件方法测量精确度低的第一个问题，RTETMeasuring 方法使用频率能够达到3579545Hz(分辨率约为0.279μs)的精确实时时钟，这对于一般的实时代码段执行时间的测量足够了。与传统的纯软件方法相同，RTETMeasuring 方法对要测量的嵌入式系统的硬件要求是，必须具有一个数字输出端口，并行和串行都可以。为了简单起见，以串口端口为例来介绍，对于其它的数字端口采用一个简单的转换电路即可。通过一根串行通信线将被测量系统的串行通信端口与一台 PC 机的串行通信端口相连。RTETMeasuring 分为两部分，一部分要嵌入到被测量代码的内部，生成状态代码；另一部分运行在 PC 机上，接受状态代码，获取此刻的精确时间并保存这些测量数据。

RTETMeasuring 方法中的状态代码由8位组成，分为两部分，高4位是被测量代码段的标识，低4位是被测量代码段开始和结束的标志。状态代码的生成代码是根据嵌入式系统实现源代码的不同，采用不同编程语言实现的函数，无论使用哪种语言，其目的都是将由参数决定的8位状态代码数据写到串行通信端口，同时还要确保状态生成和发送代码执行期间不能被中断。下面是以 C 语言为例宏定义代码：

```
# define RTET_START output(DioPort, 0x00 | id<<4)
#define RTET_STOP output(DioPort, 0x01 | id<<4)
```

这两个宏分别放在被测量代码的开始和结尾处，在这段代码开始执行前先生成状态代码并写到串行通信端口，在代码结束以后将代码段的结束状态代码写到串行通信端口。RTETMeasuring 运行在 PC 机上的部分总是监听连接到嵌入式系统的串行通信端口，当有数据到来时，读取系统的精确时钟，并将到来的数据和精确时钟值同时写入缓冲区。在测量结束后，将缓冲区中的原始数据保存到文件中。

尽管主机系统提供了十分精确的时钟，由于主机系统通常运行多个任务，系统中存在许多线程，为了提高收集到的数据的精确性，避免状态代码在串行通信端口的缓冲区中等待，应该将收集状态代码的线程优先级设置为最高。或者直接在单任务环境下，收集状态代码。

2 *操作系统调度开销及中断开销的测量方法* 要测量实时操作系统的任务调度开销，一种与文[3]中类似的简单方法是，首先创建一个有几个周期性任务组成的应用程序，这些周期性任务除了向串行通信端口输出一个固定的8位数值，什么工作也不做。这个端口就是用来收集状态代码的串行通信端口，这时，RTETMeasuring 方法运行在 PC 机上的部分收集所有这些数据，并将它们显示成时限图。

假定应用程序仅包含两个具有不同优先级的任务，低优先级的任务将会被高优先级的任务抢占，依据收集到的数据，得到的时限图如图1。

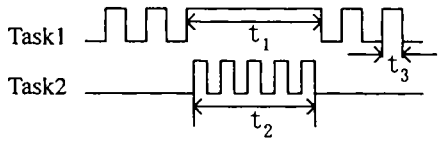

图1 测量 RTOS 调度开销图

RTOS 的调度开销可以近似为：

$$\Delta_{RTOS}=(t_1-t_2-t_3)/2 \quad (1)$$

其中 Δ_{RTOS} 为实时操作系统用于运行环境切换和调度的开销，从 t_1-t_2 中减去 t_3 的原因是在长脉冲 t_1 期间，执行了 Task1的部分代码，这部分代码产生了一个脉冲，时间为 t_3。

如果上述每个周期性任务的状态代码的生成代码段之间没有任何其它代码，则每个任务中各个脉冲的周期近似为相应的状态代码的生成代码 RTET_START 和 RTET_STOP 的开销之和，即 $2\Delta_{Macro}$，在此假定这两个开销相等。

测量中断开销可以采用相同的方法，除了在中断处理程序的开始和结束分别插入状态生成代码外，中间也要插入状态生成代码。同时，还要从采取某种手段产生中断，从而获得要测量的数据。

这几个开销的测量能够得以实现，是因为当前的 PC 机硬件一般都支持高速的串行通信，速率可达到921.6kbps，有的甚至达到1.5Mbps。但是这样的串行通信驱动程序要另编写。

2.2 计算精确执行时间的融合模型

RTETMeasuring 尽管使用了十分精确的系统时钟，与纯软件方法相同，其测量值也存在较大的误差，因为在抢占式环境中，状态代码生成和发送的开销以及 PC 机处理的开销，还有实时操作系统的任务调度开销和中断开销以不一致的方式包含在测量值中。当从 RTETMeasuring 方法得到的数据文件中读取要测量代码段的执行时间测量值时，很容易地就假定这些开销都包含在测量值之内，并且平均分配在各个任务中。通常，任务 i 的执行时间计算为 $t_{end}-t_{start}-t_{preempt}$，其中，$t_{end}$ 是宏 RTET_STOP 执行结束的时刻，t_{start} 是宏 RTET_START 执行结束的时刻，$t_{preempt}$ 是由计算得到的另外一个或

多个的更高优先级任务的执行时间。问题是这些额外的开销不可能均匀分布。

一般认为，无论何时，当高优先级的任务抢占了低优先级的任务，操作系统调度、两个宏以及 RTETMeasuring 方法 PC 机部分等的所有开销都应该分摊到低优先级任务的执行时间上。但是，与此恰恰相反，所有这些开销都与更高优先级的任务相关。如果这些开销与更高优先级的任务相关，如文[4]中对实时操作系统的任务调度开销建模的方法，对每个任务分摊的开销为$2\Delta_{RTOS}+2\Delta_{Macro}+2\Delta_{PC}$，其中两个宏 RTET_START 和 RTET_STOP 本身的开销都为 Δ_{Macro}，RTETMeasuring 方法PC 机部分的开销为 Δ_{PC}。这些开销很容易导致微秒级甚至毫秒级的误差，更甚的情况是，被多次抢占的低优先级任务的执行时间低于这些开销的总和。

因此，为了解决导致传统的纯软件方法测量精确度低的第二个问题，提出了 RTETFusion 模型，这个模型用于从根本上减少上面提到的这些开销以不一致的方式对 RTETMeasuring 方法测量值造成的误差，采用的方法是，假定所有这些开销都是常量[3]，从 RTETMeasuring 方法生成的数据文件中提取数据，进行组合为每个任务生成直接任务抢占集和直接中断集，然后利用 RTETMeasuring 方法测量得到系统开销数据和实时任务执行时间测量值与每个任务的直接任务抢占集和直接中断集相关，得到相应任务的更精确的执行时间。为了描述方便起见，定义如下几个符号：

C_i 任务 i 的最坏执行时间。

$t_{(i,j),Exe}$任务 i 第 j 次执行时，由 RTETFusion 模型计算得到的执行时间值。

$t_{(i,j),Meas}$任务 i 第 j 次执行时，由 RTETMeasuring 方法测量得到的执行时间测量值。

$S_{(i,j),Preempt}$直接任务抢占集，即任务 i 第 j 次执行时，直接抢占过它的任务的集合，$n_{(i,j),p}$是集合中元素的个数，同一个任务可以抢占它多次，每次都作为一个元素被记录，元素记为(p,q)，表示抢占任务 p 第 q 次执行。

$I_{(i,j),Preempt}$直接中断集，即任务 i 第 j 次执行时，直接中断过它的中断任务的集合，$m_{(i,j),p}$是集合中元素的个数，同一个中断任务可以中断它多次，每次都作为一个元素被记录，元素记为(p,q)，表示中断任务 p 第 q 次执行。

RTETFusion 对带有抢占过程的两个不同优先级任务执行过程建立的模型如图2所示，图中表明了抢占式环境中，RTETMeasuring 方法测量实时任务执行时间时，各个实时任务执行时间测量值、实时操作系统任务调度开销、实时操作系统中断调度开销、状态代码的生成代码开销和 RTETMeasuring 方法 PC 机端的开销之间的关系，以及这些开销对实时任务执行时间测量值的影响。

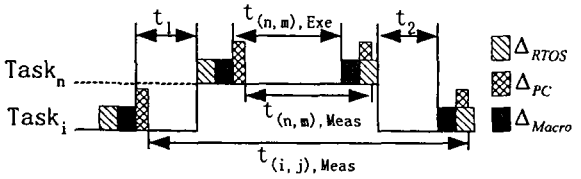

图2

图2(两任务)抢占式环境下，各种开销对实时任务执行时间测量值影响的模型图由图2可以得到任务 n 第 m 次的执行时间为：

$$t_{(n,m),Exe}=t_{(n,m),Meas}-\Delta_{Macro} \tag{2}$$

任务 i 第 j 次的执行时间为：

$$t_{(i,j),Exe}=t_1+t_2=t_{(i,j),Meas}-t_{(n,m),Meas}-2\Delta_{RTOS}-2\Delta_{Macro} \tag{3}$$

进而可以得到，多个任务抢占式环境中，任务 i 第 j 次的精确执行时间为：

$$t_{(i,j),Exe}=t_{(i,j),Meas}-t_{RTOS}-t_{TASK} \tag{4}$$

其中：$t_{RTOS}=2n_{(i,j),p}\Delta_{RTOS}+2m_{(i,j),p}\Delta_{INT}+(n_{(i,j),p}+m_{(i,j),p}+1)\Delta_{Macro}$ (5)

$$t_{TASK}=\sum_{(p,q)\in S_{(i,j),Preempt}} t_{(p,q),Meas}+\sum_{(p,q)\in I_{(i,j),Preempt}} t_{(p,q),Meas} \tag{6}$$

任务 i 的最坏执行时间为：

$$C_i=\underset{i,j}{Max}(t_{(i,j),Exe}) \tag{7}$$

由(4)、(5)、(6)、(7)式可知，RTETMeasuring 方法测量实时代码执行时间时，无需测量 PC 端的开销时间。

利用 RTETFusion 融合模型计算得到的各个实时任务或者实时代码段的执行时间，嵌入式系统的设计和开发人员可以依据设计实时规范对特定的实时任务或者实时代码段进行调整，然后再使用 RTETMeasuring 方法和 RTETFusion 融合模型进行验证，反复这个过程直到满足此时的规范要求。

2.3 分析实时性能的融合模型

测量执行时间是完全理解嵌入式系统时间特性的必要一步，获得了各部分实时代码的执行时间将有助于优化实时代码，但对于实时代码性能的分析还不够。因此，以 RTETMeasuring 方法和 RTETFusion 模型为基础，测量和计算实时代码的不同部分的执行时间，为融合模型 RTPAFusion 提供必要的数据源，从而对系统进行实时性能分析，判定特定实时任务集的可调度性。

实时性能的分析包括最终期限是否错过、周期性任务的周期是否准确和可调度性分析等。RTPAFusion 仅就可调度性分析问题，依据 RTETMeasuring 方法和 RTETFusion 模型提供的数据源，给出了一个融合模型，用于判定一个特定实时任务集，在使用固定优先级且最高优先级优先调度算法的实时操作系统环境下，是否满足可调度性要求。

可调度性分析的任务是判定一个特定实时任务集在一定的调度环境下，是否存在一个调度序列，按照这个调度序列实时任务集所有的任务都能够满足它们的最终期限，如果这个实时任务集中的任务之间存在诸如前趋约束等约束关系，这些约束关系在这个序列中也要被满足。为了简单起见，本文只考虑在固定优先级且最高优先级优先的抢占式调度环境下的实时任务集，且实时任务集中各任务之间不存在约束关系。

RTPAFusion 模型用于验证嵌入式系统是否存在潜在的时限问题的数学理论来自文[5]。这些方程整合了相关领域的许多研究成果[4,6,7]。RTPAFusion 模型是利用 RTETMeasuring方法和 RTETFusion 模型的结果，依据文[5]中的数学理论，在固定优先级且最高优先级优先的抢占式调度环境下，判定一个系统的特定实时任务集 T_{set}是否满足(8)式表达的条件[5]，当且仅当这个条件被满足时，这组实时任务集才是可调度的。

$$\min_{0<t\leqslant D_i}\left(\sum_{j=1}^{\min(i,n_{INT})}\frac{C_j+2\Delta_{INT}}{t}\left\lceil\frac{t}{T_j}\right\rceil+\sum_{j=n_{INT}+1}^{i}\frac{C_j+2\Delta_{RTOS}}{t}\left\lceil\frac{t}{T_j}\right\rceil\right)\leqslant 1,\forall i,1\leqslant i\leqslant(n_{INT}+n_{TASK}) \tag{8}$$

其中：

1)D_i 为任何一个任务的最终期限，本文假定就是这个任

务周期的结束时刻。

2)n_{INT}和n_{TASK}分别为中断的个数和任务的个数，且中断和任务统一编号，中断编号按优先级从高到低占用1…n_{INT}，任务编号也按同样的优先级顺序依次占用(n_{INT}+1)…(n_{INT}+n_{TASK})。

3)对于周期性任务，C_i和T_i分别是执行时间和周期；对于非周期性服务程序，C_i是服务程序的容量和T_i是最小的间隔周期；对于中断处理程序，C_i是执行时间和T_i是最小的间隔周期。

4)t指定了判定的时间范围，即时刻0到时刻t的这段时间内，实时任务集T_{set}中的任务是否满足它们的最终期限，即是否在本调度环境下是可调度的。

(8)式的复杂形式，在具备了各种数据信息时，实现起来并不十分复杂。如果将实时操作系统的任务调度开销和中断开销都设为常量且二者近似相等，即将中断处理程序看作非周期性任务，这时(8)式就可以简化为(9)式的形式。

$$\min_{0<t\leqslant D_i}\left(\sum_{j=1}^{i}\frac{C_j+2\Delta_{RTOS}}{t}\left\lceil\frac{t}{T_j}\right\rceil\right)\leqslant 1,$$
$$\forall\, i,\ 1\leqslant i\leqslant(n_{INT}+n_{TASK}) \qquad (9)$$

在一些处理能力有限的8位嵌入式系统中，由于几个任务的执行多为轮转方式，用(9)式判定简单，误差也不大。

3 实验及结果分析

RTETMeasuring 方法对一个实时系统五个实时任务的执行时间测量结果如表1所示，其中状态代码的第一位是任务标识，五个实时任务的标识分别为0、1、2、3、4，任务0为中断服务程序，优先级最高，后续的四个周期性任务的优先级由高到低依次为1、2、3、4；状态代码的第二位是任务开始(0)和结束(1)的标志。

表1 五个实时任务执行时间的测量结果

序号	状态代码	绝对时间(us)	序号	状态代码	绝对时间(us)	序号	状态代码	绝对时间(us)
0	10	0	14	11	5279.675	28	11	10221.204
1	11	79.339	15	20	5532.906	29	20	10473.421
2	20	344.839	16	21	5784.614	30	21	10727.096
3	21	597.664	17	30	6550.648	31	41	11274.070
4	40	1096.987	18	10	6987.697	32	30	11625.560
5	30	1824.217	19	11	7068.992	33	10	12060.199
6	10	2265.774	20	20	7779.433	34	11	12136.551
7	11	2341.202	21	00	7963.842	35	20	12826.851
8	20	3052.073	22	01	8023.223	36	00	13014.715
9	21	3301.545	23	21	8286.024	37	01	13072.693
10	10	3799.789	24	10	8737.766	38	21	13254.42
11	11	3875.497	25	11	8815.144	39	10	13740.713
12	31	4400.451	26	31	9409.166	40	11	13822.074
13	10	5204.526	27	10	10143.539	41	31	14282.636

同时，利用 RTETMeasuring 方法得到了如下的测量参数值：实时操作系统任务调度开销Δ_{RTOS}=113.941us；实时操作系统中断调度开销Δ_{INT}=50.273us；状态代码生成代码的执行时间Δ_{Macro}=20.114us 。

基于上述的测量数据，利用 RTETFusion 模型计算得到这五个实时任务的最坏执行时间，连同这些任务设计规范中的任务周期如表2所示。

表2 五个任务的周期和最坏执行时间

任务标识	周期T_i(ms)	最坏执行时间C_i(μs)
0	2.5	39.267
1	1.0	61.247
2	1.5	233.561
3	3.0	1315.211
4	12.0	2462.103

收集数据过程的时间长度为14282.636μs，由于其中包括了实时操作系统和收集数据代码的开销，所以将检查点设为12ms，即 RTPAFusion 模型中(8)式的 t=0.012。将上面的计算结果代入(8)式，进一步计算得到的结论如表3。

表3 RTPAFusion 模型对五个任务可调度性的计算结果

任务个数	可调度集	不可调度集
1	{0}{1}{2}{3}{4}	
2	{0,1}{0,2}{0,3}{0,4}{1,2}{1,3}{1,4}{2,3}{2,4}	{3,4}
3	{0,1,2}{0,1,3}{0,1,4}{0,2,3}{0,2,4}{1,2,4}	{0,3,4}{1,2,3}{1,3,4}{2,3,4}
4	{0,1,2,4}	{0,1,2,3}{0,1,3,4}{0,2,3,4}{1,2,3,4}
5		{0,1,2,3,4}

经使用逻辑分析仪的硬件方法的实验验证，表3中的结果正确。

在获取更精确测量值的模型 RTETFusion 中，本文做出了一个假设，即所有被考虑的开销都是常量。对于调度开销，这个假设只有在实时任务集中任务数比较少时才近似正确，因为调度程序要在任务队列中查找、插入和删除任务，随着任务数的增多调度开销取决于这个任务在队列中的位置。因此，对于大的实时任务集，这个融合机制要进行修正，否则判定结果存在较大误差。但是，一般的嵌入式实时系统，特别是低端的嵌入式系统，如基于8位或16位微控制器的系统，这个融合机制是适用的。

RTPAFusion 模型的判定结果的精确度取决于数据源的精确度，因此判定结果存在着一定的误差。而且，RTPAFusion 模型中(8)式中 t 的选择会直接影响任务集可调度性的判定结果，当检查点的选取略大于最慢任务的周期时，解析的误差就可以忽略：如果系统非常接近于可调度和不可调度的阈值时，解析的结果就可能错误的判定系统是不可调度的。但是，绝大多数情况下，此模型都能给出正确的结果。

结论 本文提出的嵌入式系统实时性能分析融合机制，采用了纯软件的实现方法，使用简单且代价低廉，避免了硬件方法使用复杂和成本高的困难。众所周知，在测量实时代码执行时间时，纯软件方法的通病是精确度低，针对导致这个问题的两个主要原因，借助于多源数据的信息融合思想，专门用一个融合模型 RTETFusion，将使用高分辨率系统时钟的软件方法 RTETMeasuring 获得的多源数据进行组合和相关，从根本上减少了各种额外开销以不一致方式对 RTETMeasuring 方法测量值造成的误差，从而达到满足一些嵌入式系统实时性能分析的要求。融合模型 RTPAFusion 给出了嵌入式系统实时性能分析中用于在固定优先级调度环境下可调度性分析的方法，通过本文提出的这个融合机制，固定优先级调度环境下的嵌入式系统的设计和开发人员能够查明实时代码的时限问题、优化实时代码和判定系统特定实时任

务集的可调度性及其它实时性能问题，从而提高了设计和开发的效率。由于本文在RTETFusion模型中做出了所考虑的额外开销都为常量的假设，使得本融合机制只能适用于实时任务集比较小的嵌入式实时系统。修正本文的假设，改进RTETFusion模型，使得这个融合机制能够对一个大的实时系统进行实时性能分析是进一步的研究工作。在下一步的另一个研究工作将是根据环境的要求和相应的实时调度数学理论对RTETMeasuring方法和RTETFusion模型进行调整，使得融合模型RTPAFusion能够分析其它调度环境的实时性能。

参考文献

1 Waltz E, Llinas J. 多传感器数据融合. 赵宗贵等译. 北京：机械电子部第28研究所，1993

2 White F. A model for data fusion. In: SPIE Conf. on Sensor Fusion, Orlando, FL, April, 1988

3 Stewart D B. Measuring execution time and real-time performance. In: Proc. of 2001 Embedded Systems Conf. San Francisco, CA, April 2001

4 Katcher D, Arakawa H, Strosnider J. Engineering and analysis of fixed priority schedulers. IEEE Transactions on Software Engineering, 1993, 19(9):920～934

5 Secka A. Automatic debugging of a real-time system using analysis and prediction of various scheduling algorithm implementations: [M. S. Thesis]. Dept. of Electrical and Computer Engineering, University of Maryland, College Park, MD, Nov. 2000

6 Lehoczky J, Sha L, Ding Y. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: Proc. 10th IEEE Real-Time Systems Symposium, Santa Monica, CA, Dec. 1989. 166～171

7 Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard real time environment. Journal of the Association for Computing Machinery, 1973, 20(1):44～61

（上接第156页）

殊的工具通过LambdaBus与其它工具相连。LambdaManager主要采用TCL/TK语言实现，因为TCL/TK语言具有可移植性、客户定制性等优点，同时，CORBA规范也已制定了IDL与TCL/TK语言的映射，为LambdaManager基于CORBA的构件化实现提供了可能。

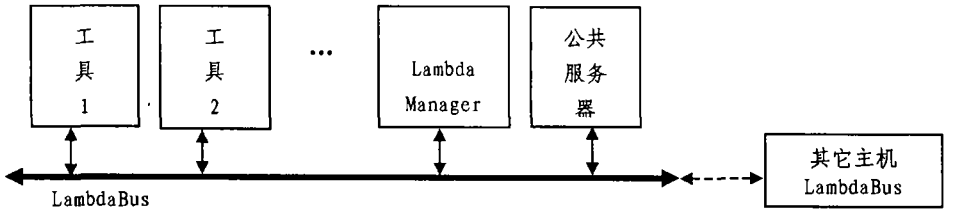

图5 LambdaBridge的总体结构

总结 由于工具总线的功能抽象集合和结构缺乏一个公认、合理的规范，以及工具通过工具总线的通信机制和策略，方便了CASE环境开发，却使得系统执行效率不高。因此，TBus体系结构还需要对松耦合与紧耦合、开放性与集成性等性能问题开展进一步的研究和实践，更好地满足系统的需求。

工具总线不但是一个软件系统，而且反映了一套系统的设计思想和软件体系结构。CASE环境采用了“工具总线＋工具构件”的体系结构[16]，一方面简化了CASE环境的结构，利于新的CASE环境的构造和已有CASE环境的重用；另一方面采用基于体系结构的构件化方法开发CASE环境，包括建模、分析、实现和演化，只要工具构件接口符合体系结构要求，工具构件的开发完全是并行的，对于CASE环境的快速开发带来了极大的便利。

参考文献

1 Shaw M. The Coming-of-Age of Software Architecture Research. In: Proc 23th Intl. Conf. on Software Engineering, Jan. 2001. 657～664

2 Liedtke J. On microkernel construction. In: Proc. of the 15th ACM symposium on operating system principles, 1995. 237～250

3 Wallnau K C, Feiler P H. Tool Integration and environment Architecture: [Technical report of SEI]. Carnegie Mellon University, 1994

4 杨芙清，邵维忠，宗志东，朱冰. 过程驱动的软件工程环境. 电子学报，1998，26(8)：24～27

5 Purtilo J M. The Polylith Software Bus. ACM Transactions on Programming Languages and Systems, 1994, 16(1):151～174

6 Bergstra J A, Klint P. The ToolBus - a component interconnection architecture. P9408, Programming Research Group, University of Amsterdam, Amsterdam, 1994

7 郭兵，沈艳，谢峻，赵平原，熊光泽. 工具总线模型研究. 已被《计算机应用》录用，2003

8 郭兵，沈艳，谢峻，赵平原，熊光泽. 工具总线：CASE环境的一种新结构. 已被《系统工程与电子技术》录用，2003

9 李保建，曾广周，林宗楷. 一种基于TriBus的软件集成框架. 计算机研究与发展，1999，36(9)

10 Medvidovic N, Khare R, Guntersdorfer M. An architecture-Centered Approach to Software Environment Integration. In: Proc. of the Ground System Architectures Workshop (GSAW 99), El Segundo, CA, March, 1999

11 Oreizy P, Medvidovic N, Taylor R N, Rosenblum D S. Software Architecture and Component Technologies: Bridging the Gap. In: Proc. of the Workshop on Compositional Software Architectures, Monterey, CA, Jan. 1998

12 Müller-Planitz C. THE CWAVE 2000 VISUAL AGENT WORKBENCH: [PhD dissertation]. Department of Computer Science, University of Utah, Aug. 2000

13 Medvidovic N, Rosenblum D S, Taylor R N. A Language and Environment for Architecture-Based Software Development and Evolution. ACM, ICSE'99, Los Angles, CA, pp. 44～53

14 Milner R, Parrow J, Walker D. A calculus of mobile process II. Information and Computation, 1992. 41～77

15 杨芙清，邵维忠，梅宏. 面向对象的CASE环境青鸟II型系统的设计与实现. 中国科学(A辑)，1995，25(5)：533～542

16 Johnson R E. Framework = (Components + Patterns). Communica-tion of ACM, Oct. 1999. 39～42