反射式实时构件的实现*)

黄 靖 卢炎生 徐丽萍
(华中科技大学计算机科学与技术学院 武汉 430074)

摘 要 反射式实时构件的实现,是对基于反射的实时构件模型规约描述机制[1]的进一步研究与实践,其实现过程中涉及到两个关键问题,即实时构件模型到实时构件实现体的转换,以及实时构件实现体反射动态性的体现。本文主要围绕这两个关键问题,给出反射式实时构件的实现方法。

关键词 反射,实时构件,模型驱动框架

The Realization of Reflective Real-time Component

HUANG Jing LU Yan-Sheng XU Li-Ping
(School of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074)

Abstract The realization of Reflective Real-Time Component is the further investigation and practice of "the Research of Timing-Constraint Component Model on the basis of Reflection Technology". During its realizing process, it involves into two key problems. That is, the transition from real-time component model to real-time component instance, and how real-time component instance reflects its dynamic state. The paper centers on these two problems as well as offering the realization methods of Reflective Real-Time Component.

Keywords Reflection, Real-time component, Model driven architecture

1 引言

文[1]中提出了一种"基于反射的实时构件模型",其能有效地规约实时应用系统中构件单元所应具有的时间约束属性和可能有的需求变化特征,支持对实时环境中动态性和适应性的表达与描述。反射式实时构件的实现,则是对基于反射的实时构件模型规约描述机制[1]的进一步研究与实践,其实现过程中涉及到两个关键问题,即实时构件模型到实时构件实现体的转换,以及实时构件实现体反射动态性的体现。其中,前一个问题是依据模型驱动框架(Model Driven Architecture, MDA)的理论与方法,由文[1]中建立的实时构件模型,导出该实现体的过程。在这个过程中,提出了一种构件模型驱动方法(Component Model Driven, CMD),用于指导反射式实时构件语法模型的代码生成,并给出构件功能语法元素和性能约束语法元素的转换规则,以及该转换工具的设计方案。后一个问题则是根据构件的反射机制,在实时环境中进行了实例测试。本文主要围绕这两个关键问题,给出反射式实时构件的实现方法。

2 相关工作

模型是对客观世界的抽象,是以精确定义的语言对系统作出的描述,其中精确定义的语言是具有精确定义的形式(语法)和含义(语义)的语言,这样的语言适合计算机的自动解释。模型驱动框架(Model Driven Architecture, MDA)[2]是OMG定义的一个软件开发框架。其关键之处在于,模型在软件开发过程中扮演了非常重要的角色,即在MDA中,软件开发过程是由对软件系统的建模行为来驱动的。其框架如图1所示。

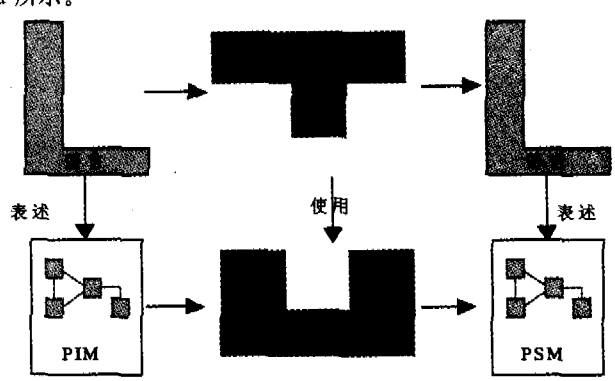


图1 基本MDA框架

框架中,模型(Model)贯穿整个MDA生命周期,是软件开发过程的驱动力。MDA生命周期主要涵盖如下3种核心模型工件。

(1)平台独立模型(Platform Independent Model, PIM):具有高抽象层次、独立于任何实现技术的模型。

(2)平台相关模型(Platform Specific Model, PSM):为某种特定实现技术量身定做,PIM会被变换成一个或多个PSM。

(3)代码:用源代码对系统的描述(规约)。每个PSM都被变换成代码。

由框架图1可示,MDA过程本质上是一个模型变换的过程,它在软件开发的生命周期中借助模型变换工具,将某个高层次的模型变换成低层次的模型,最终得到一个能在机器

*)本文得到国防预研基金项目(10104010201)资助。黄 靖 博士研究生,主要研究方向为软件工程、分布式计算技术;卢炎生 教授,博士生导师,主要研究方向为软件工程、分布式计算技术、特种数据库等;徐丽萍 副教授,主要研究方向为软件工程、分布式计算技术等。

上运行，并实际描述系统的软件模型系统。这样，提升了开发者工作的抽象层次，让开发者的注意力转移到 PIM 的设计上，减轻了他们的工作负担，大大提高了工作效率。同时，由于借助模型变换工具的作用，同一个 PIM 可以被自动变换成多个不同平台上的 PSM，从而使得 MDA 开发具有可移植性和平台间的互操作性。另外，能够让开发者更好地利用和维护设计阶段的开发文档，在高层次上实现软件复用。

3 构件模型的驱动方法

模型驱动是一种新的软件开发方法，可被视为软件工程领域中软件复用研究的一种应用实践。本文研究中，正是通过引入和借鉴 MDA 的理念及方法，来指导实时软件应用系统中的复用单元——实时构件的建模与实现，并归结为一套构件模型驱动方法（Component Model Driven，CMD），其覆盖反射式实时构件的整个开发生命周期。

在文[1]中用精确定义的 IDL/CIDL 语法和 CSP[3] 语义，分别给出了反射式实时构件的语法模型及语义模型。这里，将主要针对该构件的语法模型，研究反射式实时构件实现（体）的生成过程——CMD。其中，反射式实时构件模型是构件模型驱动方法（CMD）的驱动力，反射式实时构件的实现（体）是其产物。图 2 给出了运用 CMD 方法开发反射式实时构件的整个过程。

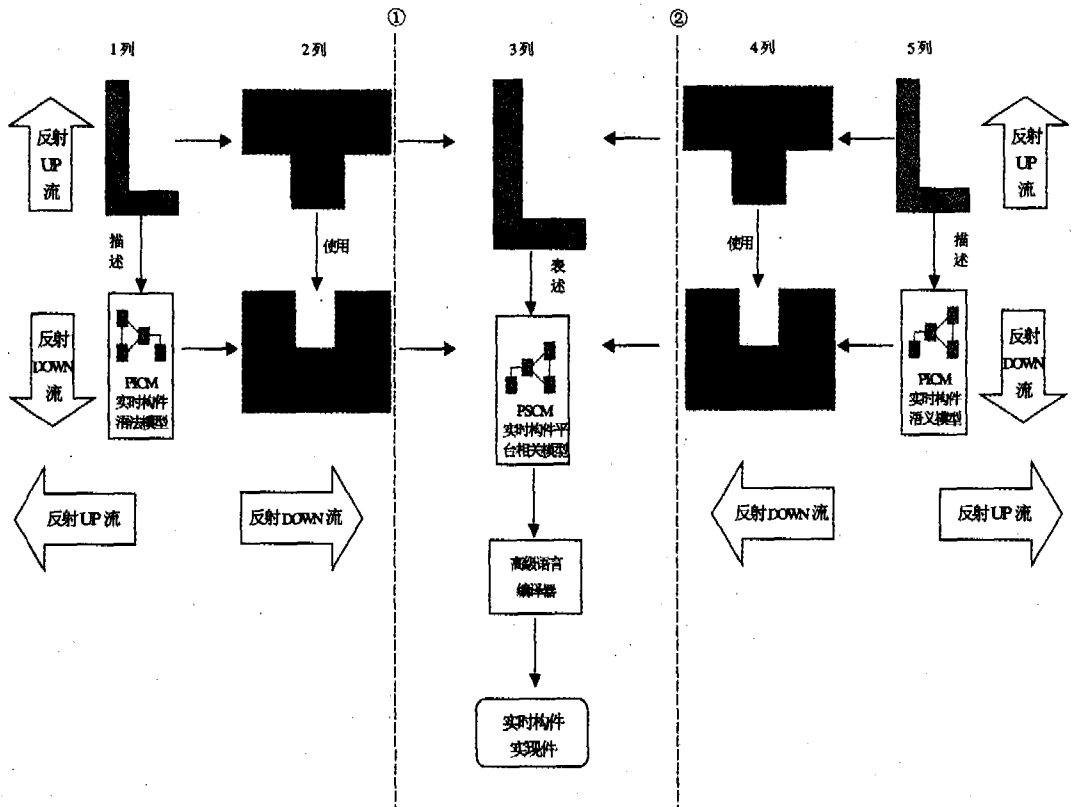


图 2　构件模型驱动方法 CMD 过程图

CMD 过程中，驱动模型主要有两大类：

（1）平台独立构件模型（Platform Independent Component Model，PICM）

它是由具有平台无关性的精确定义语言来描述构件的功能性和非功能性需求的。其又分为用扩展的 IDL/CIDL 规约的实时构件语法模型和用 CSP 表达的实时构件语义模型，这两类构件模型都是平台无关模型，具有可移植性，能有效地服务于构件复用。

（2）平台相关构件模型（Platform Specific Component Model，PSCM）

由变换工具①或②转换而成的 PSCM，是用 JAVA 等高级语言来表达的，具有一定的语言平台性。该模型作为对应高级语言编译器的输入，最终得到具有平台依赖性的构件实现（体）。

如图 2 所示，虚线①的右边部分描述了实时构件语义模型的转换过程，其过程从第 5 列经第 4 列到第 3 列可根据户用需求的变化，不断地借助反射 UP 流[1] 和 DOWN 流[1] 迭代递归模型转换信息，同时虚线①的右边部分由上至下也存在着反射流，从整体上体现了构件生成的行为反射，因而其右边部分属于一种动态模型过程；虚线②的左边部分则给出了实时构件语法模型的转换过程，其过程从第 1 列经第 2 列到第 3 列也能根据户用需求的变化，不断地借助反射 UP 流和 DOWN 流迭代递归模型转换信息，同时虚线②的左边部分由上至下也存在着反射流，从整体上描绘了构件生成中的结构反射，故此其左边部分属于一种静态模型过程。两种模型过程的转换，最终都将生成实时构件平台相关的实现体。

以往的构件实现方式，主要是通过开发者使用高级编程语言，按照某种构件规约面向底层直接编写的，如 WINDOWS 环境中 COM 构件的制作。这样，使得开发者需要花较大的精力去关注一些与构件设计无关的东西，比如构件编写的平台性因素等，从而增加了开发者的负担。而构件模型驱动方法 CMD，以构件模型为主要开发关注点，着重构件的设计，能帮助构件开发者“少流汗，多办事”，提高开发的速度和效率，以较少的努力创建更为复杂的软件系统。同时 CMD 还能根据用户需求的变化，适时地修改构件设计，并及时地、最终地反映到用户所需的构件实现（体）上来，体现了一种构

件实现(体)的动态性。这样,真正实现了构件设计时的平台无关性。

4 构件模型的转换规则

变换规则是CMD方法的重要规约原则,也是方法体系中基层(base-level)与元层(meta-level)间的关联准则。

目前CSP描述语义向高级语言或代码的自动转换,还存在着相当的难度,国内外尚未见到成型的机器转换工具,许多研究组织和团体正积极地对其进行相关研究。这里,作者的研究工作重点集中在构件语法模型的自动转换上,即图2中虚线②的左半部分。

分析文[1]中实时构件反射语法模型的特点,可将其语法模型的变换规则分为两大部分:一是常规功能性IDL/CIDL到Java的转换原则;二是经过扩展后的具有时间约束特征的IDL/CIDL到Java的转换原则。在这里我们主要介绍第二个部分的变换规则,关于第一部分可参考文[4]。

文[1]已谈到,实时反射语法构件模型中,用于标识构件时间约束特征的语法元素有构件的执行时间exec_time、执行周期exec_peri、截止期deadline以及构件的优先级pri等部分。当然,根据实际应用情况还可以对IDL/CIDL的语法规则,进行进一步的扩充。这些语法元素,作为模型里标识构件时间约束特征的自定义类型,在转换的过程中,最终将转换为高级语言代码中一个属性或一个类,即成为构件的一种性能特征元数据(metadata)。以构件的执行时间exec_time为例,下面给出转换规则的框架描述。

```
Transformation exec_time_to_Java(IDL,Java){
    params-none   //无参数变换规则
    source
      exec_time_name : exec_time;
        //源模型语言中构件执行时间变量的声明形式,跟
        在关键字source之后
    target
      exec_time_java_name : Jave_type;
        //目标模型语言中构件执行时间变量的声明形式,
        跟在关键字target之后,即转换为了一个Java
          固有变量或自定义类
    source condition
      exec_time_name in the range of interface, or outside
        the range of interface;//源模型语言条件
    target condition
      exec_time_java_name is the attribute of Java Class,or
      is a
        Interface;//目标模型语言条件
    mapping
      exec_time_name(～) exec_time_java_name;//转换
      元素体
}
```

依据变换规则框架描述的格式书写的所有构件变换规则,组成了一个变换集合,该集合中各个规则按一定的次序排列,就构成了变换定义。变换定义的具体实现,即变换工具的生成,就是下面要谈到的内容。

5 构件模型变换过程的设计与规划

构件模型变换工具,是CMD方法的核心工件,其实现是对变换定义的具体化[6]。

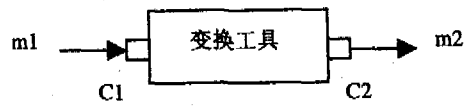


图3 构件模型变换示意图

构件模型变换过程如图3所示,其中左端left输入模型m1,右端right输出模型m2,分别对应变换工具的输入通道C1和输出通道C2。整体变换过程的CSP表示如下。

(1)变换工具黑箱时

变换工具作为黑箱的方式,是一种传统的方法,即封装变换工具的内部实现。此时,定义整体变换过程的事件、字母表、通道和进程。

定义1 输入事件in,输出事件out,变换处理事件process。

定义2 字母表 $\alpha=\{in, process, out\}$。

那么有整体转换进程 $P\triangleq in\rightarrow process\rightarrow out$。

(2)变换工具灰箱时

变换工具作为灰箱的方式,是一种反射模型,即对用户暴露变换工具的实现细节,有利于用户在适当的时候根据需要对变换工具做适当的修改。同样,先定义整体变换过程的事件、字母表、通道以及进程。

定义3 输入事件字母表 αIN,输出事件字母表 αOUT,变换处理事件字母表 $\alpha Translation$。

定义4 整体变换进程字母表 $\alpha=\{in, out\}\cup\alpha IN\cup\alpha OUT\cup\alpha Translation$。

定义5 通道1是C1,通道2是C2。其中,$\alpha c(IN)=\{mIN\mid C1.mIN\in\alpha IN\}$;$\alpha c(OUT)=\{mOUT\mid C1.mOUT\in\alpha OUT\}$,即mIN、mOUT分别为通道1的输入参数和通道2的输出参数。

定义6 输入进程 $IN\triangleq\mu X\cdot(in\rightarrow C1?\ mIN\rightarrow X)$,输出进程 $OUT\triangleq\mu X\cdot(C2!\ mOUT\rightarrow out\rightarrow X)$,以及变换处理进程Translation。

那么有整体转换进程 $P\triangleq\mu X\cdot(IN\rightarrow Translation\rightarrow OUT\rightarrow X)$,或整体转换进程 $P\triangleq\mu X\cdot(in\rightarrow C1?\ mIN\rightarrow Translation\rightarrow C2!\ mOUT\rightarrow out\rightarrow X)$。

定理1 输入参数mIN和输出参数mOUT之间存在着映射法则 f,即 $mOUT=f(mIN)$。

证明:由于mIN、mOUT分别对应着CMD方法的输入模型和输出模型,根据前一节构件模型变换规则中的论述,两者模型间存在着转换规则,那么,输入参数mIN和输出参数mOUT之间一定存在着某种映射法则 f,使得 $mOUT=f(mIN)$,并且其对应着变换处理进程Translation。即证。

因而有整体转换进程 $P\triangleq\mu X\cdot(in\rightarrow C1?\ mIN\rightarrow Translation\rightarrow C2!\ f(mIN)\rightarrow out\rightarrow X)$。

采用灰箱模式后,对变换处理进程Translation进一步细分。

定义7 $\alpha Translation=\{lex, parser, ast, code\}$,其中lex、parser、ast和code分别对应变换工具中的词法分析事件、语法分析事件、语法生成树的操作事件以及代码生成事件。

定义8 变换处理进程 $Translation\triangleq\mu X\cdot(lex\rightarrow parser\rightarrow ast\rightarrow code\rightarrow X)$。

又有整体转换进程 P≙μX·(in→C1? mIN→lex→parser→ast→code→C2! mOUT→out→X)。

由此,最终得到了整体转换进程 P 的 CSP 详尽表述。在这个过程中,用进程代数的形式细分了构件模型转换方法的每个环节,并将变换工具的内部细节暴露出来,进行模块化进程设计,有利于在设计时根据需要实现动态替换,同时也是反射式变换工具实现的设计依据及理论基础。

下面,接着讨论在变换过程的设计时,验证与支持反射性的两个性质。

性质 1 字母表相同的进程,才能进行进程间的替换,即字母表一致原则。

证明:反证法。

设进程 P 为进程 T 中一需替换的子进程,其迹为 s,再者进程 T 的迹表示为 $t=m^\frown s^\frown n$;

现假设 Q 为 P 能相互替换的待定进程,且 $\alpha P\neq\alpha Q$,其迹为 s',则替换后的进程 T 的迹 $t'=m^\frown s'^\frown n$;

由于 $\alpha P\neq\alpha Q$,则 $s\neq s'$,那么,$t\neq t'$;

迹不同,模块行为不同,则无法替换,与假设矛盾,故只有首先在字母表相同的前提下,才能进行进程间的替换,即证。

定义 9 通道相似,记作≌,当且仅当通道的流向、消息空间、参数序列一致时,才称作通道相似。

通道是进程间通信的媒介,其由流向(in 或 out)、消息空间 $\alpha c(P)=\{v|c.v\in\alpha P\}$ 以及参数序列构成,缺一不可,否则不可能形成相似的通道。

性质 2 进程的通道不相似,则不能进行进程间的替换,即通道一致原则。

证明:设有两个进程 P 和 Q,其对应的通道分别为 C1 和 C2,迹分别为 s 和 t,当

① 通道流向不同时

设 C1=in,且 C2=out,很明显 s≠t,则 P、Q 无法替换;

② 消息空间不同时

由于 αc(P)={v|C1. v∈αP},αc(Q)={v|C2. v∈αQ},且 αc(P)≠αc(Q),根据性质 1 可得 P、Q 无法替换;

③ 参数序列不同时

参数序列不同,使得 s≠t,则 P、Q 无法替换。

综合上述,在 C1≌C2 的情况下,才能进行进程间的替换,即证。

6 实验与测试

为检验实时构件运行时环境反射式动态调整的性能,在这里我们采用 ACE+TAO 开发包,并在 Linux OS 2.6.1,512M RAM,1.8GHz Intel CPU 的单机上进行开发与测试。首先设置 3 个实时构件,并采用弹性实时调度算法[7],在调度中分别为其分配 3 个调度线程,构成任务集,如表 1。

表 1 测试用例

GUID	关键性	重要性	弹性系数	周期范围	初始周期	执行时间
1	1	1	2	[8, 16]	8.000	4.000
2	2	1	1	[7, 9]	7.000	3.000
3	3	1	0	[6, 8]	6.000	2.000

6.1 典型执行时间(TET)的反射动态调整测试

为便于观察,此时任务集中只设置一个构件任务,即最差执行时间(WCET)=6.000s,周期=[2s, 6s]。在 20s 内的采样区间内,典型执行时间(TET)的收敛过程如图 4 所示,从图中可以看出,即使在任务到达时,作业的 WCET 和实际情况相差很大,在运行过程中,上层调度模块也可使其迅速向实际值收敛(新值占 0.5 权重)。这一特性还使得在软实时应用时,用户不必去测试或模拟作业的最差执行时间,这样有利于简化软实时构件的开发,因为准确测量作业的最差执行时间殊非易事。

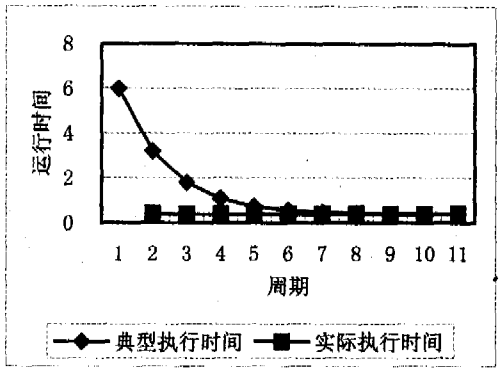


图 4 TET 收敛过程

6.2 周期任务反射动态调整测试

该测试主要考察在系统超载时,有些构件的时间约束特征不满足请求时,弹性调度算法上层调度模块能否及时作出反应,压缩任务集的 CPU 利用率,以使任务集回到可调度的状态。与此类似,在系统负载较低时,上层调度模块能否及时提高各任务的 CPU 利用率,以提高服务质量,充分利用资源。系统负载和任务周期变化的关系见图 5(a),为便于分析,同时给出了系统负载和作业典型执行时间的变化对比。从图中可以看出,系统超载的根本原因是构件任务 3 的加入,而构件任务 3 是刚性负载,所以在 8.1 时刻(相对时间)系统超载后,构件任务 1 和构件任务 2 的周期即被扩展,系统的负载相应回落到 $U_{hb}^{MLTF}=1$ 以下。

另外,从图 5(b)可以看到,作业的实时执行时间会发生剧烈起伏,幅度将近 50%,这与 CPU 的 cache 是否命中有关,尽管对于硬实时应用来说,不能乐观地将 cache 命中时的执行时间作为 WCET,但对于软实时应用来说,显然应该采用过去一段时间内作业的平均执行时间,即典型执行时间,而非 WCET。另一方面,该图也显示了在任务实施模块提高 cache 命中率的重要性,这可以大幅提高系统吞吐量,因此有必要对任务集、CPU 个数与 cache 命中率之间的关系进行定量分析,以作为 Leader/Follower 线程池 Follower 排队模式的理论基础。

结束语 构件实现(体)是构件模型在相关硬件平台的具体实现,是基于构件软件开发(Component-based Software Development, CBSD)的组成单元,是一种有平台相关性二进制级别的软件单位。反射式实时构件实现体,凭借自身的特点,能有效地组建开发实时应用系统,满足对软件应用系统有时间约束特征的客户群。

参考文献

1 黄靖,卢炎生,徐丽萍. 基于反射的实时构件模型规约描述研究. 计算机科学,2006(10)

2 鲍志云译. 解析 MDA. 北京:人民邮电出版社,2004

3 Hoare C A R. Communicating Sequential Processing. Prentice Hall, 1985

(下转第 254 页)

抽象事务存储模型的最基本特点和组成部分，并扩展单核处理器模拟器 SimpleScalar，我们已经完成了基于循环并行化的支持事务存储 OpenCMP 模拟器的方案设计和模拟器的实现。基于 OpenCMP 模拟器，我们用 fft 做了一个简单的性能评测的例子来验证模拟器的设计。实验表明，OpenCMP 已能够较好地支持 CMP 事务存储模型的研究。

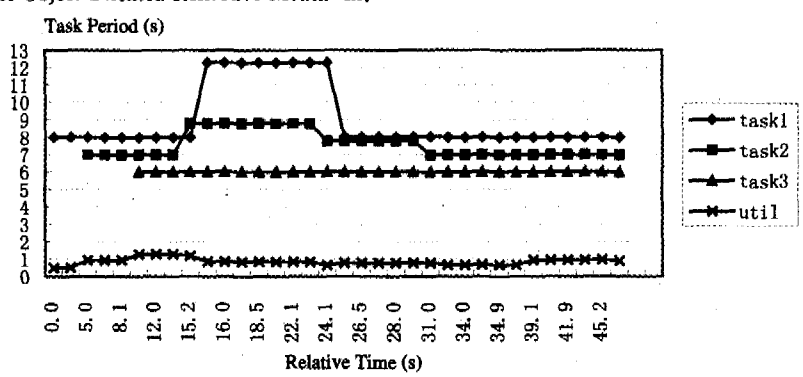
我们对 OpenMP 的进一步改进主要从以下几个方面展开：(1)加入对于子程序调用和对于传统的程序并行化方法的支持。(2)进一步优化事务存储模型的几个抽象模块的设计，以提高性能。目前，对于总线的模拟，由于要进行多次的访问，模拟性能不高。(3)支持对于全系统的模拟，包括对操作系统等的模拟。(4)完善模拟器的性能评估模型。(5)由于目前采用手工编译插入指令，对于工具集中编译器没有进行扩展，不能进行大规模程序的并行化。为了进行更大规模的评测，需要修改工具集中的编译器提供对更多更大的 benchmark 程序的模拟分析。此外，目前的控制软件开销过大，也是需要改进的地方。

参 考 文 献

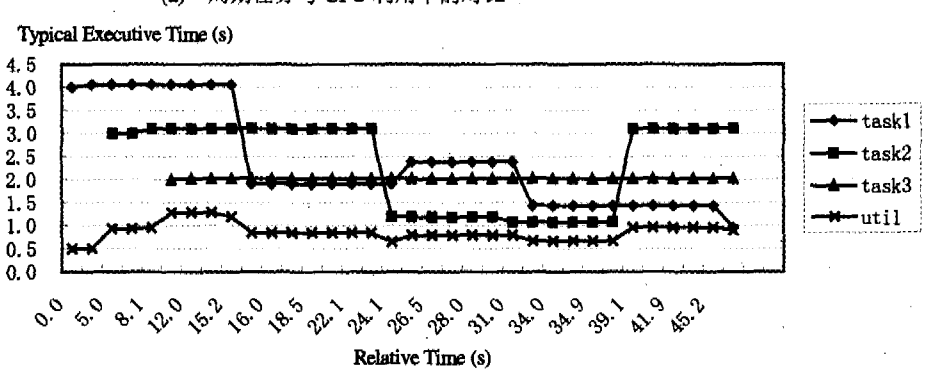
1 Tullsen D M, Eggers S J, Levy H M. Simultaneous multithreading: Maximizing on-chip parallelism. 22nd Annual International Symposium on Computer Architecture, June 1995
2 Marr D T, Binns F, Hill D L, et al. Hyper-threading technology architecture and microarchitecture. Intel Technology Journal, Feb. 2002
3 Diefendorff K. Power4 Focuses on Memory Bandwidth. Microprocessor Report, Oct. 1999
4 Clabes J, et al. Design and implementatin of the power5 microprocessor. In ISSCC Digest of Technical Papers, Feb. 2004
5 Burger D, Austin T M. The SimpleScalar tool set version 2.0:[Technical Report]. 1342. Computer Sciences Department, University of Wisconsin, June 1997
6 Austin T, Ernst D. SimpleScalar Tutorial (for release 4.0). http://SimpleScalar.com/docs/simple_tutorial_v4.pdf
7 Martin M M K, Sorin D J, Beckmann B M, et al. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. Computer Architecture News (CAN), September 2005
8 SESC: http://sesc.sourceforge.net/index.html
9 Nathan L B, Erik G H, Steven K R. Network-Oriented Full-System Simulation using M5. The Sixth Workshop on Computer Architecture Evaluation Using Commercial Workloads, Feb. 2003
10 Mendel R, Edouard B, Scott D, et al. Using the SimOS Machine Simulator to study Complex Computer Systems. ACM TOMACS special issue on Computer Simulation, 1997
11 路放，安虹，梁博，等. OpenSMT：一个同时多线程处理器模拟器的设计和实现. 计算机科学，2006(1)
12 Sohi G, Breach S, Vijaykumar T. Multiscalar Processors. Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA-22), June 1995
13 Herlihy M, Moss J B. Transactional Memory: Architectural Support for Lock-Free Data Structures. Proceedings of the 20th Annual International Symposium on Computer Architecture, May 1993
14 Knight T. An architecture for mostly functional languages. Proceedings of the 1986 ACM Conference on LISP and Functional Programming (LFP '86), August 1986
15 Hammond L, Wong V, Chen Mike et al. Transactional Memory Coherence and Consistency. Proceedings of the 31st Annual International Symposium on Computer Architecture, June 2004
16 Hammond L, Carlstrom B D, Wong V, et al. Programming with Transactional Coherence and Consistency (TCC) ASPLOS04, October 2004
17 Ananian C S, Asanovic K, Kuszmaul B C, et al. Unbounded Transactional Memory. In: Proceedings of the Eleventh International Symposium on High-Performance Computer Architecture. Feb. 2005
18 Moore K E. Thread-Level Transactional Memory. Wisconsin Industrial Affiliates Meeting. Wisconsin Industrial Affiliates Meeting, Oct. 2004
19 Rajwar R, Herlihy M, Lai Konrad. Virtualizing Transactional Memory. Proceedings of the 32nd Annual International Symposium on Computer Architecture. Jun. 2005
20 Hammond L, Hubbert B, Siu M, et al. The Stanford Hydra CMP. IEEE MICRO Magazine, March-April 2000

(上接第 222 页)

4 Object Management Group. The CORBA Architecture and Specification, Reversion 3.0. OMG Inc, March 2004
5 Cazzola W. Evaluation of Object-Oriented Reflective Model. In: Proceeding of ECOOP Workshop on Reflective Object-Oriented programming and Systems(EWROOPS'98), Brussels, Belgium, July 1998



(a) 周期任务与 CPU 利用率的对比



(b) 典型执行时间与 CPU 利用率的对比

图 5