

一种面向 M5 的图形化并行模拟技术研究

计卫星,王永辉,宋 红

(北京理工大学 计算机学院,北京 100081)

摘 要: 针对目前使用 M5 模拟器时出现的两个主要问题,提出一种面向 M5 的图形化配置与并行测试方法,通过可视化界面实现被测试系统中功能部件的连接和属性配置,并自动生成运行 M5 所需的 Python 脚本,同时支持多核处理器上多个测试程序的并行运行,提高了 M5 模拟器的使用效率。

关键词: 模拟器;并行计算;多核处理器;计算机系统结构

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2010)08-0177-04

Study of a Visualized and Parallel Simulation Method for M5

JI Wei-xing, WANG Yong-hui, SONG Hong

(School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

Abstract: In this paper, we present a graphical tool and a parallel simulation method for M5. Designers can configure the target architecture efficiently using this visualized tool, which automatically generate the Python scripts according to the interconnection between components. It also supports multiple benchmarks configuration and simulation. In view of this, it improves the efficiency of system simulation when using M5.

Key words: simulator; parallel computing; multi-core processor; computer architecture

1 引言

M5^[1]是美国密歇根大学 2001 年发布的一款开源模拟器,该模拟器使用 C++ 完成底层核心模块的开发,而同时选择 Python 完成上层系统接口部分的实现,编译执行和解释执行两种语言的巧妙结合既使 M5 获得了较高的模拟性能,也使系统结构设计者在多次修改系统连接关系和属性配置后无须对模拟器核心模块进行重复编译,提高了工作效率,并具有较好的可扩展性和可配置性。目前 M5 既支持用户级模拟,也支持全系统级模拟;特别是对多处理器及网络模拟的支持较为突出。近年来随着多核处理器的不断推广及对其研究的不断深入,M5 日益受到学术界的重视,越来越多的研究人员在实验中采用 M5 作为其系统性能模拟和验证的工具。

然而 C++ 和 Python 两种语言的使用在为 M5 带来较高性能和灵活性的同时,也使模拟器的学习和使用更加困难。如果要对新的系统结构进行模拟

与验证,使用者必须熟练掌握两种语言,并熟悉模拟器内部二者的结合与交互方式。Python 语言是一个很年轻的语言,相比于其他语言 Python 的用户群不够壮大,能够熟练运用 Python 的系统结构设计者相对较少,因此无形中提高了 M5 使用的入门门槛。目前比较流行的另外一个模拟器 Simics^[2]也采用编写脚本进行模拟器配置的方法。此外,多核处理器已在桌面系统和服务器中得到广泛应用,但是限于模拟本身的特点,目前 M5 仍然无法利用多线程技术实现模拟加速^[3]。

针对 M5 使用过程中出现的上述两个主要问题,文中提出一种面向 M5 的图形化辅助工具,支持被测试与评估的目标系统结构进行图形化的系统连接与配置,并能够根据系统构成部件之间的连接关系,自动生成 M5 运行时所需的 Python 脚本;此外该图形化接口充分利用和发挥多核处理器的性能优势,支持同一测试集中多个测试程序的并行测试、监控与结果收集。通过以上两种方式,极大简化了系统

结构设计者的工作,提高了 M5 的使用效率,缩短了研究者对特定系统结构进行模拟验证所需的时间。

2 图形化系统结构配置

目前 M5 中支持的 CPU 有两种,分别是 SimpleCPU 和 O3CPU,其中 SimpleCPU 仅支持功能模拟,而 O3CPU 支持乱序执行、超标量及流水线的时序级模拟^[4]。内存系统的表示也分为两大类:分别是存储器部件和互连部件,其中存储器部件包括缓存、主存和 I/O 设备等,而互连部件包括总线和互连网络。M5 的内核采用了事件驱动的仿真技术,并使用面向对象方法实现了系统中各个功能模块的封装与分离。例如 CPU、Cache、存储器、总线和 I/O 部件等,在底层实现中封装为各个类,并定义了类之间

通讯的通用接口。通过参数设定对象实例(部件)功能和行为,并通过通用接口之间的连接实现复杂系统的搭建。这样的实现方式不但使 M5 具有较好的可扩展性,也提高了设计者对系统进行扩展与集成的灵活性。

与 M5 内部功能模块相对应,本文给出的图形化设计环境中提供了各个功能部件和互连部件,用户通过拖拽的方式选取各种功能部件并完功能部件的基本布局,然后通过互连线将各个功能模块进行互连,形成被模拟系统的互连关系。系统整体布局与互连关系构建完成后,可单击任何一个部件,对其参数进行配置,例如 Cache 大小,Cache 行大小,访问延迟等;CPU 的类型,频率等。与 M5 的实现相对应,目前支持的功能模块如表 1 所示。

表 1 支持的功能模块

M5 中的类	对应功能模块	说 明
DerivO3CPU	DerivO3CPU	支持乱序执行的 CPU 模型
InOrderCPU	InOrderCPU	支持顺序执行的 CPU 模型
TimingSimpleCPU	TimingSimpleCPU	支持存储系统时序模拟的 CPU 模型
AtomicSimpleCPU	AtomicSimpleCPU	支持原子存储访问的 CPU 模型
BaseCache	BaseCache	Cache
	DCBaseCache	数据 Cache
	ICBaseCache	指令 Cache
Bus	Bus	总线
PhysicalMemory	PhysicalMemory	物理内存

其他需要配置的参数包括:检查点、系统输入、系统输出、测试程序等。完成上述配置之后,即可生成相应的 Python 脚本进行模拟,同时将模拟器输出信息重定向输出到信息输出面板。

如图 1 所示为使用该辅助工具实现的系统结构图。另需说明的是,目前 M5 的 Python 脚本中可根据传入的参数完成分支选择,实现不同的系统结构配置,在可视化环境中,加入条件分支理论上是可行的,但是会使设计看起来比较混乱。因此在目前的实现中并没有加入相应的支持,用户可以通过快速复制并修改系统结构图的方式实现多个系统结构设计的配置与模拟。在设计与评估过程中,任何时刻用户都可对目前的系统结构配置及构件之间的互连关系进行保存,我们将每一个系统结构设计图保存为一个 XML 文档,用户可在后续的工作中装入并进行修改。由于 M5 可在 x86、SPARC、Alpha、PowerPC 等处理器和 Linux、MacOS X、Solaris、OpenBSD、Cygwin 等系统中运行,为保证可视化辅助工具的可

移植性,面向 M5 的可视化界面使用 JGraphx 实现,JGraphx 是一个 mxGraph 版本的 Java Swing 库,提供了强大的可交互图形功能,不仅简化了 M5 模拟器可视化的设计,也保证了该辅助工具的可移植性。

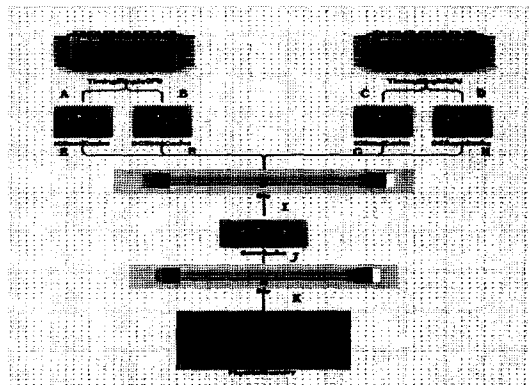


图 1 系统结构图举例

M5 模拟器在实现了可视化配置后,用户工作量极大降低,主要体现在以下两个方面:

(1) 可视化配置与代码编写相比更加直观,最大限度降低由于手工编写代码引入错误的可能性,用户可将主要精力放在系统结构的设计上;

(2) 除修改已有模块或增加新的模块之外,用户不需要熟悉 M5 内部结构及两种语言的结合与交互方式,即可使用 M5 模拟器进行验证与设计。

3 Python 脚本自动生成

与 M5 的结构相对应,整个脚本生成分为三部分,包括被测试系统中模块基本配置信息对应的 Python 类的生成、模块互连关系的生成,以及 Checkpoint、Sampling 及 Warm-up 等设置信息的生成。

为了与 M5 中各个类相对应,对于系统结构图中的每个节点需要生成一个相应的 Python 类,该类的基类由该节点的类型及配置信息决定,在生成过程中给该类确定一个唯一的名称,并根据基类和用户配置的信息生成相应的类代码。为了实现系统部件之间的连接,需要生成部件之间连接关系的 Python 代码。由于部分功能模块具有多个端口,且每个端口有特殊的命名方式和特定的连接方向,所以在生成连接关系之前,需要对部件的连接端口进行标注。例如确定 Cache 的两个端口是 cpu_side 还是 mem_side 的,其中 cpu_side 靠近 CPU 端,mem_side 靠近物理内存端。节点端口标记与代码生成算法描述如下:

(1) 检查每个 CPU 节点,将与 CPU 相连的节点的端口标记为 cpu_side;

(2) 从物理主存节点开始;将与物理主存相连的节点的端口标记为 mem_side;

(3) 遍历其他所有节点,如果其一个端口已被标记为 cpu_side,则将该节点的另一端口标记为 mem_side;如果已被标记为 mem_side,则将该节点的其他端口标记为 cpu_side;

(4) 遍历其他所有节点,如果节点任一端口均未标记,则查看与每一个端口相连的所有其他部件的端口,如果与特定端口相连的所有其他端口均已标记,则根据这些节点的标记结果对该端口进行标注;

(5) 重复步骤(3)和(4)直到所有节点和边被标记;

(6) 根据标记结果和连接关系生成相应 Python 代码。

初始状态时,所有端口及连线均未标记,该算法从每个 CPU 节点开始自上而下对节点和连线进行

标记,标记相应连线并生成 CPU 与这些节点之间的连接关系;同时从物理内存开始自下而上标记所有连线和节点。最后根据标记结果和连接关系生成对应的代码。

以图 1 中的系统结构图中的一个处理器节点为例,为生成其对应的 Python 类,首先获取其类型,通过将类型名和 ID 组合的方式确定类名。这里假设第一个 CPU 的类名为 TimeSimpleCPU0,超类名为 TimeSimpleCPU。然后为 TimeSimpleCPU0 类添加属性和值,例如我们将该 CPU 的时钟频率从“1GHz”修改为“2GHz”,则类生成相应的语句“clock = '2GHz'”。通过标记算法,便完成了系统功能模块互连关系对应代码的生成。生成的代码片段如下所示:

```
cpu _ TimingSimpleCPU0. icache = cache _ ICBBaseCache2
cpu _ TimingSimpleCPU0. icache _ port = cache _ ICBBaseCache2. cpu _ side
cpu _ TimingSimpleCPU0. dcache = cache _ DCBaseCache3
cpu _ TimingSimpleCPU0. dcache _ port = cache _ DCBaseCache3. cpu _ side
cpu _ TimingSimpleCPU1. icache = cache _ ICBBaseCache4
cpu _ TimingSimpleCPU1. icache _ port = cache _ ICBBaseCache4. cpu _ side
cpu _ TimingSimpleCPU1. dcache = cache _ DCBaseCache5
cpu _ TimingSimpleCPU1. dcache _ port = cache _ DCBaseCache5. cpu _ side
phy _ mem _ PhysicalMemory9. port = bus _ Bus8. port
cache _ BaseCache7. mem _ side = bus _ Bus8. port
cache _ BaseCache7. cpu _ side = bus _ Bus6. port
cache _ ICBBaseCache2. mem _ side = bus _ Bus6. port
cache _ ICBBaseCache4. mem _ side = bus _ Bus6. port
cache _ DCBaseCache3. mem _ side = bus _ Bus6. port
cache _ DCBaseCache5. mem _ side = bus _ Bus6. port
```

最后根据用户设置的 Checkpoint、Sampling & Warm-up 等参数生成其他配置代码;将上述参数设置都存储在 Python 生成的一个名为“options”的字典中,由于字典中数据的存储都是以(键,值)存储,这样其它需要访问运行时参数的脚本只需要访问 options 变量。

4 并行模拟

M5 模拟器运行时只能使用单个核进行系统模拟,而一般的测试集中包含多个测试程序,测试不同程序在相同系统上的运行性能可能相当耗时。随着多核时代的到来,当前主流计算机系统都具有多个处理器,只使用单核进行模拟测试,造成系统资源

的极大浪费.因此,可以利用图形化界面实现多个测试程序并行运行,充分发挥多核系统的性能优势.

在自动生成 Python 脚本时,可通过图形化界面,让 M5 用户根据需要进行多个测试程序,根据系统中可用的处理器数量,同时生成多个进行 Python 脚本,并启动多个 M5 运行实例实现测试程序的并行模拟,既充分利用本地系统资源,又提高了工作效率.此外,将多个测试程序的输出信息同时显示在图形化界面,提供了直观的对比,使得 M5 用户可以更好的分析所设计系统的性能.

文中使用双核系统对多个测试程序的并行模拟进行了简单测试,实验中 M5 采用用户级模拟模式,设定的模拟平台为 ALPHA.从韩国 SNU(Seoul National University)^[5]实时研究组的 SNU-RT 测试程序集中挑选了 3 个典型的程序,对并行测试环境进行了分析.这 3 个程序分别为 bs、fir 和 crc.bs 是一个实现 15 个整型元素数组二分检索算法的程序;fir 是实现了有限冲激响应和高斯函数的程序;crc 是一个循环冗余校验示例程序.测试结果如表 2 所示.

表 2 串行模拟与并行模拟比较

测试组	基准测试程序	测试程序耗时(s)	串行耗时(s)	并行耗时(s)	加速比
1	bs	0.61	1.64	1.02	1.61
	fir	1.03			
2	crc	1.79	2.82	1.84	1.53
	fir	1.03			
3	bs	0.61	2.40	1.81	1.32
	arc	1.79			

从表 2 中可以看出,在多核处理器上采用并行模拟的方法可以显著缩短模拟所需的时间,从而提

高了工作效率.

5 结束语

文中针对 M5 模拟器使用中出现的两个主要问题,提出一种对系统结构进行图形化配置与验证的测试方法,该方法能够根据功能部件之间的互连关系及属性配置自动生成 M5 运行所需的 Python 脚本.并可根据用户的配置和系统中可用的处理器数量,同时生成多个 Python 脚本,实现对多个测试程序的并行模拟.通过上述两种方法减少了系统结构设计者的工作量,提高了 M5 的使用效率.下一步将对该图形化辅助工具进行完善,使其能够生成针对不同模拟器的测试脚本,例如实现对 Simics 的支持.

参考文献:

- [1] Nathan L Binkert, Ronald G Dreslinski, Lisa R Hsu, et al. TheM5 simulator: modeling networked systems[J]. IEEE Micro, 2006, 26(4): 52-60.
- [2] Peter S Magnusson, Magnus Christensson, Jesper Eskilsson. Simics: a full system simulation platform[J]. Computer, 2002: 50-58.
- [3] 许建卫, 杨伟, 潘晓雷, 等. 体系结构模拟器的技术和发展[J]. 信息技术快报, 2008, 6(1): 1-25.
- [4] 时磊, 逢珺, 杨磊, 等. M5 模拟器的内核分析及应用[J]. 微计算机应用, 2009, 30(4): 30-34.
- [5] SNU. SNU real-time benchmarks[EB/OL]. [2009-12-17]. <http://archi.snu.ac.kr/realtime/benchmark/>.

作者简介:

计卫星 男, (1980-), 博士, 讲师. 研究方向为计算机系统结构及并行计算.

(上接第 176 页)

参考文献:

- [1] Gildea D, Jurafsky D. Automatic labeling of semantic roles[J]. Computational Linguistics, 2002, 28(3): 245-288.
- [2] Thompson CA, Levy R, Manning CD. A generative model for semantic role labeling[C]// Proceedings of ECML-2003, LNAI 2837. Berlin, Heidelberg Springer, 2003: 397-408.
- [3] Che W, Zhang M, Liu T, et al. A hybrid convolution tree kernel for semantic role labeling[C]// Proceedings of ACL-2006. Sydney, 2006.
- [4] Baker C F, Fillmore C J, Lowe J B. The berkeley

frameNet project[C]// Proceedings of ACL & Coling-1998. Canada: Quebec, 1998: 86-90.

- [5] Darroch J N, Ratcliff D. Generalized iterative scaling for log-linear models[J]. Annals of Mathematical Statistics, 1972, 43(5): 1470-1480.
- [6] 刘挺, 车万翔, 李生. 基于最大熵分类器的语义角色标注[J]. 软件学报, 2007, 18(3): 565-573.

作者简介:

于江德 男, (1971-), 博士, 副教授. 研究方向为自然语言处理、信息抽取、文本数据挖掘等.