片上系统并行建模的软件开发方法研究

全盛程 王 勇
(桂林电子科技大学计算机科学与工程学院 桂林 541004)

摘 要 作为片上系统描述的语言库,SystemC允许定义可执行的硬件虚拟平台原型且已成为工业上的标准,但其串行模拟内核的固有属性仍无法充分利用对称多处理器(SMP)的计算能力。针对该问题,提出一种从底层内核引擎到高层建模的软件开发方法。该方法通过并行编程改进内核调度算法以实现真正的并行模拟内核,并结合接口方法调用(IMC)机制和SystemC的分层通道概念,提出一种事务级快速开发的软件开发体系。实验表明,利用该方法建模片上系统可提高模拟速度,且在工程上有较高的开发效率。

关键词 片上系统,虚拟原型,内核调度,SystemC,事务级建模
中图法分类号 TP311.52 **文献标识码** A

Research of Software Development Method for the Parallel Modeling of System-on-chip

QUAN Sheng-cheng WANG Yong
(School of Computer Science & Engineering, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract As the description language library of System-on-chip, SystemC can define executable virtual prototype of hardware platform and has become de facto industrial standard, but can not make full use of the computation capacity of SMP (Symmetrical Multi-Processing) for the inherent property of sequential simulated kernel. To solve the problem, this paper proposed a software development method from kernel engine to high level modeling. By utilizing parallel programming, the method improves kernel scheduler algorithm and implements really parallel simulation kernel, on the basis of which, combining interface-method-call mechanism and hierarchy channel conception of SystemC, proposes a kind of rapid software development hierachy. The experiment shows modeling system-on-chip by the method can efficently improve simulated speed, and possesses highly developed efficiency in project.

Keywords System-on-chip, Virtual prototype, Kernel scheduler, SystemC, Transation level modeling

1 引言

现代片上系统设计流程要求芯片设计和软件开发同时进行,以缩短产品推出时间,在此形势下提出虚拟原型[7]的概念。利用系统描述语言在更高的抽象层对芯片进行事务级建模不仅可以对架构进行仿真验证,更可以利用系统仿真工具协助软件开发和调试。SystmeC[1]作为C++库能够从寄存器传输级(RTL)到事务处理级(TLM)的不同抽象级别上对系统进行建模,并由于可以利用面向对象语言的强大功能而成为开发仿真器的最佳选择。在工业和学术上开发的辅助建模、模拟和调试的框架大多以SystemC为主体[4]。

随着多核处理器片上系统(MPSoc)复杂度的提升及其内部模块数量的增加,芯片的功能和所支持的软件业务也趋于复杂和多样化,这些都会导致仿真器的模拟时间急剧膨胀,可见建立高性能仿真平台意义重大。模拟时间增加的重要原因是系统级建模语言和模拟平台都是典型单线程的串行执行,阻碍仿真器充分利用对称多核处理器的并行计算能力[11]。基于这些原因,本文通过并行编程技术改进SystemC的调度机制,使得进程在时钟精确级别和高层事务级都可以并发执行,在充分利用对称多处理器的计算能力提高模拟速度的基础上保持代码兼容性。同时充分利用接口方法调用的加速机制和分层通道[9],提出模块管道概念在事务处理级提供一个封装良好的框架以简化建模机制和加快开发速度。

本文第2节简要介绍片上系统并行模拟的发展现状,然后说明OSCI SystemC内核引擎机制,最后详细介绍本文提出的并行内核引擎;第3节介绍整个建模框架的架构方法、理念和编程模式;第4节介绍实验环境,分别在串行和并行内核上,通过不同的进程和模块数进行性能分析。

2 并行模拟内核

2.1 相关工作

目前国内外工业和学术界为提高SystemC模拟速度,从寄存器传输级到事务级提出较多改进内核的方法。Daniel Gracia Pérezd[2]等人通过移除无用的、不需要唤醒的逻辑进程[7]加速模拟,但是需要显示指明各种信号的依赖性致使代码复杂度增加。Rafik S. Guindi[3]通过进程切片方式解决此

到稿日期:2011-10-11 返修日期:2012-02-18 本文受国家自然科学基金项目(61163058)资助。
全盛程(1987—),男,硕士生,主要研究方向为计算机网络,E-mail:cn86qsc@gmail.com;王 勇(1964—),男,博士,教授,主要研究方向为计算机网络技术与应用、信息安全、计算智能等,E-mail:wang@guet.edu.cn(通信作者)。

问题，但其仍运行于串行内核。Mello [4] 和 Viaud [6] 通过实现并行离散模拟事件[12]达到加速模拟，但是逻辑进程要显示与线程绑定从而导致某些线程存在大量空闲时间，进程之间传递大量时间消息容易造成死锁及系统资源的浪费，其内核实现的编程接口无法满足工程应用需求。在 Combes[5] 中提出几种改进模拟内核的方式，但都失去与 SystemC 代码的兼容性且无法成熟地应用于工业开发。

2.2 OSCI SystemC 内核

典型 SystemC 应用程序被描述为结构部分和行为部分[5]。以芯片内部相互连接的组件为基础，结构部分被转化为各种通过通道互联的模块(sc_modules)，模块内部通过实例化子模块来支持层次性结构。对应于物理进程，行为部分通过逻辑进程实现，并通过敏感事件列表来控制进程调度。每个模块内部可以包含多个逻辑进程。

图 1 说明了 OSCI SystemC 的调度算法。如果用户不特殊指明，初始化阶段所有进程被设置为可执行状态。进程在评估阶段被串行执行，如果立即事件通知发生，所有敏感于此事件的进程将被设置为可执行态。在此阶段中，所有可执行的进程都运行完毕后，则进入更新阶段。在评估阶段中，若进程向通道中写入新值会产生更新请求，内核执行在更新阶段执行请求并由此产生 Δ 事件和时间事件。通过此形式，在下一个 Δ 周期到来之前，评估阶段不需要立即考虑通道中的新值。完成更新阶段后，触发所有对 Δ 事件敏感的进程，使得模拟时间在每个 Δ 周期都推进 Δ 时间同时重新进入评估阶段执行循环。当没有 Δ 事件触发时，进入时间更新阶段，取出最早的时间事件，触发所有对此事件敏感的进程重新回到评估阶段。如果到达模拟的最终时间，整个模拟过程结束。

```
initialize(); /* 初始化阶段 */
while true do
  for all handles in runnable do /* 评估阶段 */
    execute(handle);
  update() /* 更新阶段 */
  /* Δ事件通知阶段 */
  if exist_delta_event() then
    trigger_all_delta_events();
    continue;
  /* 时间事件通知阶段 */
  t := next_simulation_time();
  if Tend== t then exit();
  else trigger_timed_events(t); continue;
```

图 1 SystemC 内核调度算法

2.3 并行模拟内核

SystemC 时钟是通过两个动态的 SC_METHOD 进程互相触发推进时间前进，在某些事务级建模中不会用到此类进程，并行内核中将时钟的进程更改为 SC_THREAD 类型。OSCI SystemC 在任何时间点只能执行一个进程，因此所有的逻辑进程都是串行执行。通过并行编程技术更改目前的内核，使其在任何时间点都可并行执行进程，以充分利用对称多处理器的多核运算能力和并发执行的功能。整个模拟时间会随着 CPU 内核的增加而线性下降。更改后的内核使得△通知事件和时间通知事件是并行的同步压入堆栈，从而增加了进程执行顺序的不确定性。

并行模拟内核调度算法如图 2 所示。调度器通过创建线程池产生并发执行环境，线程池中线程数目与 CPU 核数相当。所有的线程都可以互相独立地调度逻辑进程实现并行执行且独立地产生各种事件，由主线程继续维护更新阶段和事件通知阶段。在逻辑进程并发执行阶段，所有线程从可执行进程堆栈中取得句柄。为保持数据的一致性，在并发执行进程时需要进行同步，对于需要注册的通道，在立即事件触发后同样需要加入到执行堆栈的敏感进程句柄，以及各种事件队列的读写操作，利用系统互斥体进行同步操作。

```
/* * * * * * 主次线程共用的调度函数 * * * * * */
Procedure: parallel_excute_process()
  var method_h, thread_h;
    while method_h := pop_runnable_method() do
      /* 执行 SC_METHOD 进程 */
       method_h->semantics();
    /* 执行 SC_THREAD 进程 */
    sc_cor_pkg::yield(next_cor ());

/* * * * * * 主线程算法 * * * * * */
create_thread_pool();
initialize(); /* 初始化阶段 */
while true do
  trigger_all_threads();/* 触发等待中的所有调度线程 */
  parallel_excute_process();
  /* 逻辑进程执行完毕，调度线程重新挂起 */
  wait_all_threads_to_end();
  update(); /* 更新阶段 */
  if exist_delta_event() then/* △事件通知 */
    trigger_all_delta_events(); continue;
  /* 时间事件通知阶段 */
  t := next_simulation_time();
  if Tend== t then break;
  else
    trigger_timed_events(t); continue;

/* * * * * * 线程池内的所有线程调度算法 * * * * * */
while true do
  wait_for_trigger(); /* 等待主线程触发进行调度 */
  parallel_excute_process();
  /* 通知主线程已经调度完毕 */
  send_signal_to_mainthread();
```

图 2 并行内核调度算法

在线程池创建时，可以利用操作系统默认的方式进行线程调度，本文利用操作系底层函数[14]将线程和 CPU 核进行一一对应绑定以实现最优化处理[13]。逻辑进程没有与特定调度线程绑定，这可以充分利用线程不断循环执行逻辑进程来减少线程的空闲时间，从而加速模拟。

3 并行建模体系

3.1 事务处理级框架

SystemC TLM2.0 专门为建模存储器映射的片上总线设计，以实现不同厂商模型互相兼容[8]。其提出发起者、目标者以及连接它们的套接字概念，同时引入接口方法调用机制加速模拟速度[1]，但此机制却屏蔽了 SystemC 基础库中良好的

通道模式。其通道分为基本通道和分层通道[9]，是模块相关接口的实现类，通过面向接口的编程方式实例化相互关联模块的端口导出接口实现互联，体现良好的物理连接概念。但是通道的运用需要互联模块两端各建立进程来进行读写，从而降低模拟速度。本文结合两者，提出模块通道（module_piper）的概念，使得模块连接通过通道以及数据的发送通过接口来调用机制。

内存管理可以通过基本的编程方式简单实现，TLM2.0为内存管理的兼容性所提供的复杂通信协议不适合快速掌握和开发。在松散定时建模和近似定时建模中，需要设置事务相位进行适当延时和相应事件的触发，这会增加编程的复杂度并且降低仿真性能。而类似网络处理器建模，内部模块数据被适当处理从而产生一定的延时后发送出去，无需过多相位表示，针对此类情况设计事务处理级应用框架。模块的连接由模块通道完成，用户只需把数据和延迟时间加入到通道中即可，所有的发送工作由通道内部自动完成。

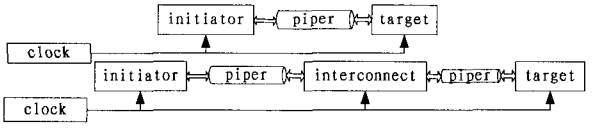

图3　模块连接模式

```
/******发起者模块******/
SC_MODULE(initiator){;
  /*通道指针*/
  module_piper(initiator,target,req_struct,rsp_struct)* m_piper;
  void my_handle(rsp_struct & trans) ;/*响应提取函数*/
  /*注册响应提取函数*/
  m_piper -> reg_rsp_interface(this, & initiator::my_handle);
  /*向通道加入请求数据*/
    m_piper -> req_into_pipe(st_req); }};

/******目标者模块******/
SC_MODULE(target){
  /*通道指针*/
  module_piper(initiator,target,req_struct,rsp_struct)*
  m_piper;
  void my_handle(req_struct& trans);/*请求提取函数*/
  /*向通道注册请求提取函数*/
  m_piper->reg_req_interface(this, & target::my_handle);
/*向通道加入响应数据*/
m_piper -> rsp_into_pipe(st_rsp); }};

/******顶层函数******/
int sc_main(…){
  module_piper (initiator, target, req_struct, rsp_struct)
  piper("pipe");/*模块通道*/
  initiator initia(…,&piper);
  target tar(…,&piper);/*为模块内部通道指针赋值*/
  piper.join();}/*通道两端建立连接*/}
```

图4　模块通道连接方式

模块的连接方式如图3所示。通道是由一个module_piper模板类实现，内部组合各种注册类、套接字和数据管理类。注册类将特定的模块方法注册进通道，使该方法可以从通道中取得相应模块发送过来的数据，此模块方法分别称为请求和响应提取方法。套接字利用新的请求接口（class tlm_req_if）和响应接口（class tlm_rsp_if）建立发起者和目标者的连接。数据管理类通过继承TLM2.0提供的peq_with_get类，在类的内部建立动态逻辑进程实现数据的自动发送功能。这些具体的实现属于内部的细节，用户只需声明模块管道，向管道内注册提取函数，然后向管道加入数据和提取数据，便可很好地屏蔽数据的发送策略，使得用户更专注于模块功能实现。图4显示单一的发起和目标者的连接方式，模块的建立和连接很好地解耦，其他的互联模式以及多对多模式的情形类似。

3.2　软件体系

整个软件体系从底层到应用层都做到优化处理，充分地利用硬件性能以及结合高级编程理念，在保证线性提升模拟速度的前提下简化建模过程。调度线程和特定的CPU核绑定可互相独立地调度逻辑进程，没有采用逻辑进程和特定线程绑定使各个调度线程可以无空闲地调度逻辑进程。高层的架构屏蔽TLM2.0复杂的通信协议和内存管理机制，从一个更抽象的层面建模。

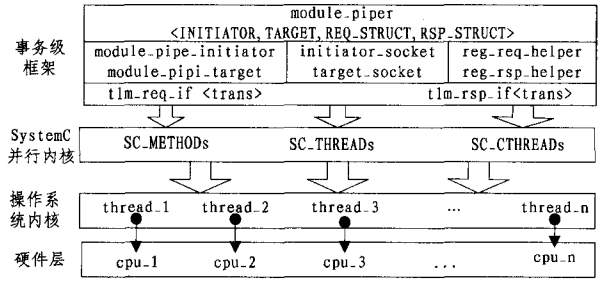

图5　片上系统的并行建模软件体系

4　实验以及结论

实验目的是比较串/并行SystemC内核运行性能及验证并行模拟框架的稳定性。为比较性能，分别在仿真器运行周期数一定而逻辑进程数变化和逻辑进程数一定而运行周期变化的情况下，测试两种内核的仿真时间（s）和每秒钟执行的周期数（cycle/s）。为测试稳定性，仿真器创建大量的逻辑进程和模块，每个发起者、目标者至少各有8逻辑进程，这样在进程数高达1000情况下大约有100个模块。操作系统是windows平台，编译器为VC8.0，测试用机的CPU分别是2核、8核和16核。

图6表明随着运行线程数量的增加，串行的模拟内核运行的时间迅速上升，而随着CPU核数的增加，并行的内核执行时间上升变慢。图8表明线程数量一定的情况下，随着模拟周期数的增加，串行内核执行时间增加幅度远超过并行内核。综合图7和图9，串行内核随着CPU核数的增加，模拟速度和运行性能的提升幅度很低，但是并行内核的模拟速度随着CPU核数的增加线性提升。理想情况下，假设CPU有n个核，并行模拟内核的执行速度接近串行的n倍，但由于涉及到外设的串行访问、程序的同步等问题，此速度无法完全达到。程序在长时间运行时，没有出现内存泄露或溢出等异常中断情况，且串、并行的执行效果一致。并行建模体系能够很好地利用对称对处理器的计算能力来加速模拟，而且简化模块的连接方式以及数据的传输模式更容易对片上系统内部模块建模。

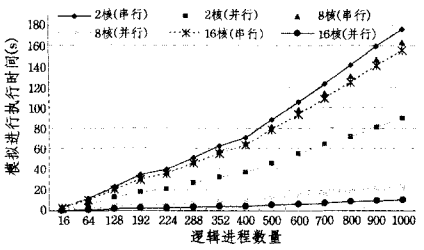

图 6 串/并行内核的执行时间

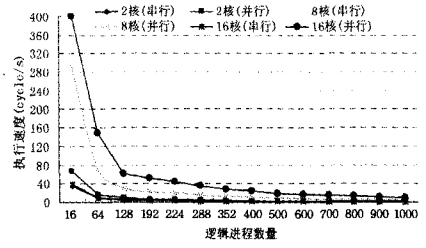

图 7 串/并行内核执行速度

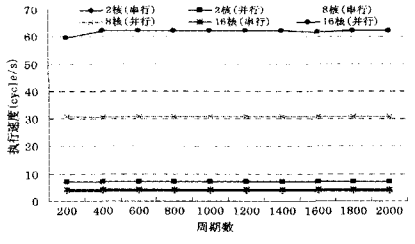

图 8 周期变化下执行时间

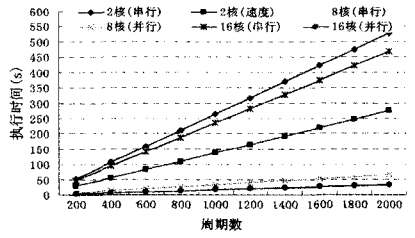

图 9 周期变化下的运行速度

结束语 并行模拟内核主要利用多核 CPU 并行计算能力,以及操作系统并行执行线程的机制改进调度策略来并行执行逻辑进程,其依然保留了串行内核的离散事件模拟(DES)特性,从而导致在主线程同步事件通知和更新事件的过程中其他线程被挂起而耗费模拟时间。为进一步提升仿真性能,并行离散事件模拟(PDES)算法代替目前的模拟方式,时钟模型也由单一的改为分布式的,每个逻辑进程都拥有相应的本地时钟,进程间通过带有时间信息的数据来同步时钟以及传递信号。这样,所有的调度线程就可以不停地扫描循环可执行的逻辑进程并运行,进一步减少由于事件同步和通知导致线程挂起的时间。

在事务级建模中,可以充分利用 TLM 所提供的发起者和目标者的概念,提供并封装相应的接口简化建模方式。在并行离散模拟内核情况下,需要做一部分封装内核的工作以为事务级提供一致的接口,这样在事务级的代码就可以完全向后兼容了。

参 考 文 献

[1] Open SystemC Initiative[EB/OL]. http://www.systemc.org

[2] Pérez D G, Mouchard G, Temam O. FastSysC: A Fast SystemC Engine[C]// Design, Automation & Test in Europe Conference & Exhibition (DATE). Paris, 2004

[3] Guindi R S, Naguib Y N. SplitPro: A Tool to Over- come SystemC Scheduling Inefficiencies[C]// Microelectronics (ICM), 2010 International Conference. Cairo, 2010: 347-350

[4] Mello A, Maia I, Greiner A, et al. Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations[C]// Design, Automation & Test in Europe Conference & Exhibition (DATE). Dresden, 2010: 606-609

[5] Combes P, Caron E, Desprez F, et al. Relaxing Synchronization in a Parallel SystemC Kernel[C]// The 2008 IEEE International Symposium on Parallel and Distributed Processing With Applications. Sydney, 2008: 180-187

[6] Viaud E, Pecheux F, Greiner A. An Efficient TLM/T Modeling and Simulation Environment Based on Conservative Parallel Discrete Event Principles[C]// Design, Automation & Test in Europe Conference & Exhibition (DATE). Munich, 2006: 1-6

[7] Chandy K M, Misra J. Distributed simulation: A Case Study in Design and Verification of Distributed Programs[J]. IEEE Trantions on Software Engineering Software, 1979, 5(5): 440-452

[8] 李挥,陈曦. SystemC 电子系统级设计[M]. 北京:科学出版社, 2010

[9] 陈曦,徐宁仪. SystemC 片山系统设计[M]. 北京:科学出版社, 2003

[10] Bhasker J. SystemC 入门(第 2 版)[M]. 夏宇闻,甘伟,译. 北京:北京航空航天大学出版社, 2008

[11] Ezudheen P, Chandran P. Parallelizing SystemC Kernel for Fast Hardware Simulation on SMP Machine[C]// 23rd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation. New York, 2009: 80-87

[12] Fujimoto R M. 并行与分布仿真系统[M]. 李革,刘宝宏,张耀程,译. 北京:电子工业出版社, 2010

[13] Stallings W. 操作系统:精髓与设计原理(第 6 版)[M]. 陈向群,陈渝,译. 北京机械工业出版社, 2010

[14] MSDN library[EB/OL]. http://msdn.microsoft.com/library

(上接第 304 页)

[2] Linn C, Debray S. Obfuscation of Executable Code to Improve Resistance to Static Disassembly[C]// Proceedings of the 10th ACM Conference on Computer and Communications Security. New York: ACM, 2003: 290-299

[3] Gough K J, Cifuentes C, Corney D, et al. An experiment in mixed compilation/interpretation[C]// Proceedings of the Fifteenth Australian Computer Science Conference. Hobart: Australian Computer Society, 1992: 315-327

[4] Cifuentes C, Van Emmerik M, Ramsey N, et al. Experience in the Design Implementation and Use of A Retargetable Static Binary Translation Framework [R]. CA, USA: Sun Microsystems, 2002

[5] 曾鸣,赵荣彩,姚京松. 一种基于重定位信息的二次反汇编算法[J]. 计算机科学, 2007, 34(7): 284-287

[6] 蒋烈辉,陈亮,等. 基于控制流和数据段分析的反汇编策略研究[J]. 计算机工程, 2007, 33(2): 94-96

[7] 胡刚. 固件程序代码逆向分析关键技术研究[D]. 郑州:郑州信息工程学院, 2010

[8] 张翠艳. 固件代码安全缺陷分析技术研究[D]. 郑州:郑州信息工程学院, 2010