第42卷 第6期 西 安 交 通 大 学 学 报 Vol.42 №6
2008年6月 JOURNAL OF XI′AN JIAOTONG UNIVERSITY Jun. 2008

多核并行测试系统研究

王磊，肖明清，夏锐
（空军工程大学工程学院，710038，西安）

摘要：针对传统测试程序在并行测试中效率提升不明显的问题，结合测试流程提出了基于任务、数据、数据流的3种分解方法，以解决测试中的可并行性分解问题.将大幅增加了并行能力的多核(MCP)技术引入并行测试平台，使得任务并行度的实时性进一步得以提升，进而提出了基于工作量的并行测试任务调度策略，优化了MCP平台的测试流程与任务调度，解决了单核测试程序对MCP技术利用率不高的问题.实验结果表明，基于MCP的测试平台较之单核平台的测试效率可提升30%～50%.

关键词：多核；并行测试；测试程序；并行调度策略
中图分类号：TP301 **文献标志码**：A **文章编号**：0253-987X(2008)06-0683-05

Study on a Parallel Test System Based on Multi-Core

WANG Lei, XIAO Mingqing, XIA Rui
(Engineering College, Air Force Engineering University, Xi′an 710038, China)

Abstract: The efficiency of the parallel test is limited with the traditional test program. Three decomposed methods, which are based on missions, data and data flows, are brought forward to solve the problem of the parallel decomposition in test processes. The multi-core technology is introduced into the field of the parallel test, and it improves the real-time capability of test task with its parallel ability. The schedule strategy based on the task amount is advanced to optimize the test flow and the task schedule of MCP system, and solves the problem that single-core program couldn't utilize MCP technology effectively. The experiment results show that the performance of the plat based on MCP is enhanced about 30%-50% against the single-core plat.

Keywords: multi-core; parallel test; test process; parallel schedule strategy

并行测试技术根植于并行处理技术，是指自动测试系统(ATS)在同一时间内完成多项测试任务，包括完成多个被测单元(UUT)的测试，在单个UUT上同步或者异步运行多个测试任务，以及对UUT多项参数的测量等[1].单核处理器多线程技术(SCMT)是通过提高计算速度或者利用人类感知能力来模拟多任务环境[2]，这显然不适用于实时性要求较高的并行测试.为了解决此问题，一般采用多处理器结构，但会大幅增加测试成本.

多核(MCP)技术是指在单个物理处理器内集成多个核，它不仅可有效降低测试的复杂度和成本，而且能在单个处理器内支持基于硬件、线程级的并行处理[3-6]，但目前测试任务的分解和并行调度的复杂性限制了该项技术的应用.

当前，并行任务调度的研究引起了学术界关注，相关调度算法也应运而生，但这些算法仍然受限，如仅适用于计算资源无限的情形[7]、仅考虑了计算资源而未涉及测试资源冲突[8]、未涉及并行性[9]等.为了有效解决多核平台在并行测试中的问题，本文对多核技术应用于并行测试系统进行了理论论证，研究了任务分解的方法并提出了基于工作量的任务调度策略.

收稿日期：2007-06-27. 作者简介：王磊(1984—)，男，博士生；肖明清(联系人)，男，教授，博士生导师. 基金项目：总装“十一五”重点预研资助项目(51317030103).

1 MCP并行测试结构

1.1 多核测试平台并行测试架构

MCP是在单个处理器芯片内实现两个或多个“执行核”[10]，执行核自身具有执行集合及体系结构资源，每个核均通过仲裁器与前端总线接口相连，仲裁器通过同步功能模块来完成各核之间的通信.

在多核平台上，各线程是在独立的执行核上并行运行的[11]. 设计多核测试平台应集中于设计、开发、设置应用程序的多线程，协调各线程及其操作之间的关系等. 目前，多核测试平台的瓶颈是软件，大多数测试程序都是基于串行模式编写的，并不能发挥MCP的高并行处理能力[3].

1.2 测试程序分解

MCP并行测试平台的软件体系结构如图1所示.

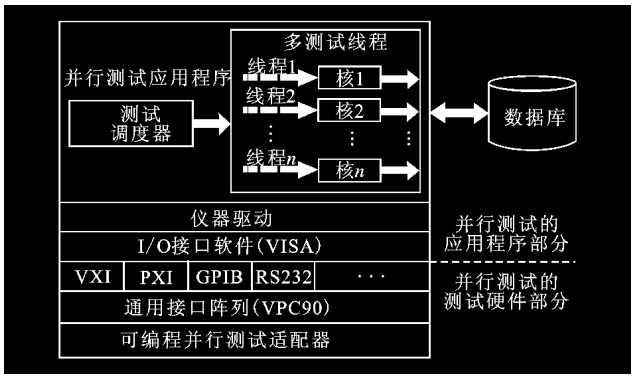


图1 多核并行测试平台的软件体系结构

在MCP上，通过充分考虑以多线程方式执行的任务及重构算法之后，可将测试流程分解成多个独立的任务，并实现同步运行. 本文根据测试的任务、数据处理和测试流程，将分解方式分为任务分解、数据分解、数据流分解，如表1所示.

表1 测试流程分解方式

分解方式	说明	应用
任务分解	根据执行功能分解	分解不同的程序行为，如数据采集和绘图等
数据分解	根据对不同数据的操作进行分解	用于各种信号处理，如音频或图像等
数据流分解	根据数据在任务间的流动分解	将一个线程输出，并作为另一个线程的输入

任务分解是一种能够简单实现并行执行的方法，如果两个测试任务能够同时运行，就可以对其进行调度，从而形成二者之间的并发执行. 对测试数据进行操作，包括绘图、计算和判断等，采用数据分解和数据流分解是最有效的方法.

2 并行测试任务的调度策略

2.1 并行测试任务描述

在并行测试系统中，测试经常由多个任务构成，而且任务之间关系复杂. 并行测试的结构可用图论中的有向无环图(DAG)描述，也可用一个二元组 $G=(W,E)$ 表示，其中 $W=\{(w_{11},w_{12},\cdots,w_{1j}),(w_{21},w_{22},\cdots,w_{2j}),\cdots,(w_{i1},w_{i2},\cdots,w_{ij})\}$，$E=\{e_1,e_2,\cdots,e_i,\cdots\}$. 在DAG中可用节点模拟 w_{ij}，表示第 i 个UUT的第 j 项测试，w_{ij} 的工作量可用其权值 x_{ij} 表示. 同样，在DAG图中可用有向边模拟 e_i，表示任务之间的准备时间和依赖关系，任务之间的边也可由 (w_1,w_2) 来表示，称 w_1 为 w_2 的前驱任务，而用 (w_1,w_2) 的权值 $c(w_1,w_2)$ 表示任务之间的准备时间.

如果某一测试共有4个UUT，每个UUT分3步测试，且测试的准备时间与任务的控制关系集合 $K_{\text{Task}}=\{w_{13}\prec w_{12},w_{12}\prec w_{11},w_{23}\prec w_{22},w_{22}\prec(w_{11},w_{21},w_{31}),w_{33}\prec w_{32},w_{32}\prec w_{31},w_{43}\prec(w_{42},w_{32}),w_{42}\prec(w_{31},w_{41})\}$（$w_a\prec w_b$ 表示 w_a 受控于 w_b），则其DAG如图2所示.

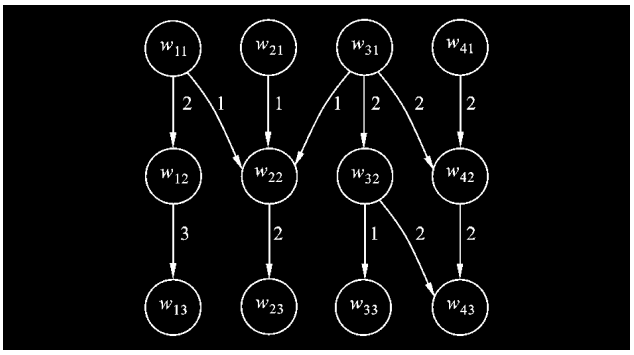


图2 4个UUT分3步测试的DAG

2.2 假设条件与相关定义

假设如下：i 为测试中UUT的个数，j 为测试任务最多UUT的测试任务数；测试在任意时刻、任意资源只能处于一种状态；测试任务占用资源的时间等于该项任务的测试时间；每个测试任务都在有限长的时间内释放其占用的资源；各个测试步骤的测试时间和测试准备时间是已知的.

定义1 每个测试子任务 w_n 的测试时间为 t_n，准备时间为 $t'_n=\max(c(w_m,w_n))$（w_m 为 w_n 的前驱任务），则 $T_n=t'_n+t_n$ 为测试量，而

$$\boldsymbol{P}^{i\times j}=\begin{pmatrix}t_{11}&\cdots&t_{1j}\\ \vdots&\ddots&\vdots\\ t_{i1}&\cdots&t_{ij}\end{pmatrix}$$

为测试时间矩阵. 如果 UUT 测试任务数不足 j,则矩阵中相应位置补 0.

定义 2　要完成测试子任务 w,就必须先对 $w_1,w_2,\cdots,w_n$ n 项子任务进行测试,而 $w_1,w_2,\cdots,w_n$ 为 w 的前驱任务,$k_{ij}=n$ 为前驱任务数,则

$$\boldsymbol{K}^{i\times j}=\begin{pmatrix}k_{11}&\cdots&k_{1j}\\ \vdots&\ddots&\vdots\\ k_{i1}&\cdots&k_{ij}\end{pmatrix}$$

为前驱任务矩阵.

定义 3　设 w 为一项测试子任务,而受控于它的测试子任务为 $w_1,w_2,\cdots,w_n$,测试时间分别为 $t_1,t_2,\cdots,t_n$,测试准备时间为 $T_1,T_2,\cdots,T_m$,则 $x=\sum_{i=1}^{n}t_i+\sum_{j=1}^{m}T_j+T$ 为测试任务 w 的工作量,而

$$\boldsymbol{X}^{i\times j}=\begin{pmatrix}x_{11}&\cdots&x_{1j}\\ \vdots&\ddots&\vdots\\ x_{i1}&\cdots&x_{ij}\end{pmatrix}$$

为工作量矩阵. 如果有些 UUT 不足 j 步测试,则矩阵中相应位置补 0.

定义 4　测试过程中用到的所有资源的集合 $R_{\text{Task}}=(r_1,r_2,\cdots,r_s)$ 为资源集,每个测试子任务 w_{ij} 的资源集为 $R_{\text{Task}ij}=(r_1,r_2,\cdots,r_n)$,$n\in s$.

2.3　多融合矩阵

一般情况下,在并行测试中不同的测试任务可能会用到同样的测试资源,即发生了测试资源冲突问题. 为了防止资源竞争而导致死锁,应对每个正在测试的子任务 $w_1,w_2,\cdots,w_n(1\leqslant n\leqslant(i\times j))$ 建立资源融合矩阵 $\boldsymbol{H}_{w_n}^{i\times j}$. 在 w_n 的测试资源 $R_{\text{Task}n}=(r_1,r_2,\cdots,r_n)$ 中,没有相同资源的测试过程时,其矩阵值为 1,其余为 0.

当任务的并行度大于 2 时,$\boldsymbol{H}_{w_n}^{i\times j}$ 只能表示任务 w_n 的可融合测试. 融合矩阵相应地变为多融合矩阵 $\boldsymbol{H}^{i\times j}=\boldsymbol{H}_{w_1}^{i\times j}\otimes\boldsymbol{H}_{w_2}^{i\times j}\otimes\cdots\otimes\boldsymbol{H}_{w_n}^{i\times j}$,其中⊗表示矩阵对应位置的元素逻辑与. $\boldsymbol{H}^{i\times j}$ 中为 1 的测试任务是可以运行的.

2.4　基于工作量的并行调度策略

一般情况下,同时满足可测条件的测试任务不止一个. 基于时间最短的任务优先调度未考虑后续测试,它属于局部最优的调度策略. 就此问题,本文提出了基于工作量的并行调度策略,描述如下.

(1)计算

$$\boldsymbol{X}^{i\times j}=\begin{pmatrix}x_{11}&\cdots&x_{1j}\\ \vdots&\ddots&\vdots\\ x_{i1}&\cdots&x_{ij}\end{pmatrix}\left(x_{ij}=\sum_{i=1}^{n}t_i+\sum_{j=1}^{m}T_j+t'_{ij}+t_{ij}\right)$$

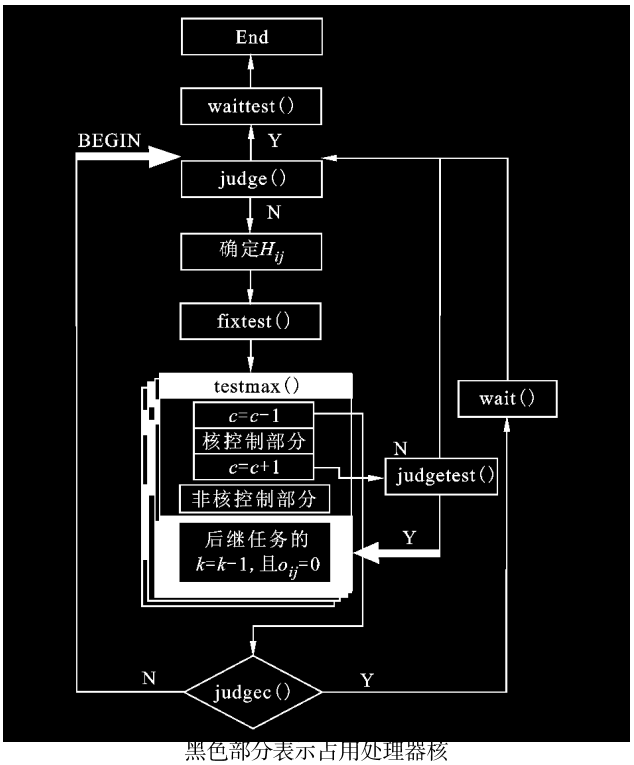
$$\boldsymbol{K}^{i\times j}=\begin{pmatrix}k_{11}&\cdots&k_{1j}\\ \vdots&\ddots&\vdots\\ k_{i1}&\cdots&k_{ij}\end{pmatrix}$$

建立测试遍历矩阵

$$\boldsymbol{O}^{i\times j}=\begin{pmatrix}1&\cdots&1\\ \vdots&\ddots&\vdots\\ 1&\cdots&1\end{pmatrix}$$

来标记测试任务是否完成,其中 1 表示测试未完成.

(2)设变量 c 的起始权值为 CPU 核的数量,表示测试系统的并行度,占用核权值则减 1,释放核权值则加 1. 对每一测试过程还应设变量 k 为前驱任务数,系统每完成一项测试则其后续任务的 k 值减 1,当 k 为 0 时,进入待测队列. 并行测试任务调度策略如图 3 所示.



黑色部分表示占用处理器核

图 3　并行测试任务调度策略

在并行测试任务调度策略中,若函数 judge()判断 $\|\boldsymbol{O}^{i\times j}\|$ 为 0,则表明测试任务分配完成,此时函数 waittest()等到所有测试项目完成后退出;否则,根据正在运行的测试求 $\boldsymbol{H}$,用函数 fixtest()求取 $\boldsymbol{X}'=\boldsymbol{X}\otimes\boldsymbol{H}\otimes\boldsymbol{O}$,得出可运行测试子任务 w_n 为 $\boldsymbol{X}'$ 相应

位置不为 0 且 $k_{ij}=0$ 的测试任务. 同样,用函数 testmax()创建新的线程运行工作量权值最大的测试任务,并修改相应权值.

需要指出的是,测试任务并不会全程占用处理器核,多核多线程程序可使一个处理器核同时处理多个测试任务,因此在核控制部分前后应该分别加 $c=c-1$ 和 $c=c+1$ 语句. 在 $c=c-1$ 语句之后执行 judgec(),以判断是否存有核闲置,并开始新的测试; 在 $c=c+1$ 语句之后执行 judgetest(),以判断开始的测试是否有核需求.

按照上述的调度策略对所有测试任务进行调度,直至测试结束.

3 多核平台实例性能分析

以图 2 为例,采用基于 VXI 总线的测试设备,利用 VXI-1394 的控制方式,与 CPU 采用单核或双核的主控计算机分别连接,进行单核单线程顺序测试、单核多线程并行测试与多核多线程并行测试实验. 为了便于说明问题,在编程过程中让测试过程全程占用处理器核.

(1)设在 $R_{\mathrm{Task}}=(r_1,r_2,r_3,r_4)$ 中,资源分别代表程控电源、AD 卡、视频采集卡等,各测试任务资源使用情况如表 2 所示.

表 2 测试任务的资源使用

测试任务	UUT1	UUT2	UUT3	UUT4
测试 1	r_1	r_3	r_2	r_1, r_4
测试 2	r_2	r_1, r_3	r_4	r_1
测试 3	r_2, r_3	r_3	r_2	r_4

(2)从单步测试中得出数据

$$\boldsymbol{P}=\begin{bmatrix}1&2&2&1\\3&3&2&1\\2&2&3&2\end{bmatrix}$$

且 $c=2$.

求

$$\boldsymbol{X}^{i\times j}=\begin{pmatrix}x_{11}&\cdots&x_{1j}\\\vdots&\ddots&\vdots\\x_{i1}&\cdots&x_{ij}\end{pmatrix}=\begin{bmatrix}19&10&25&8\\10&8&12&7\\5&4&4&4\end{bmatrix}$$

$$\boldsymbol{K}_{ij}=\begin{bmatrix}0&0&0&0\\1&3&1&2\\1&1&1&2\end{bmatrix}$$

并建立遍历矩阵

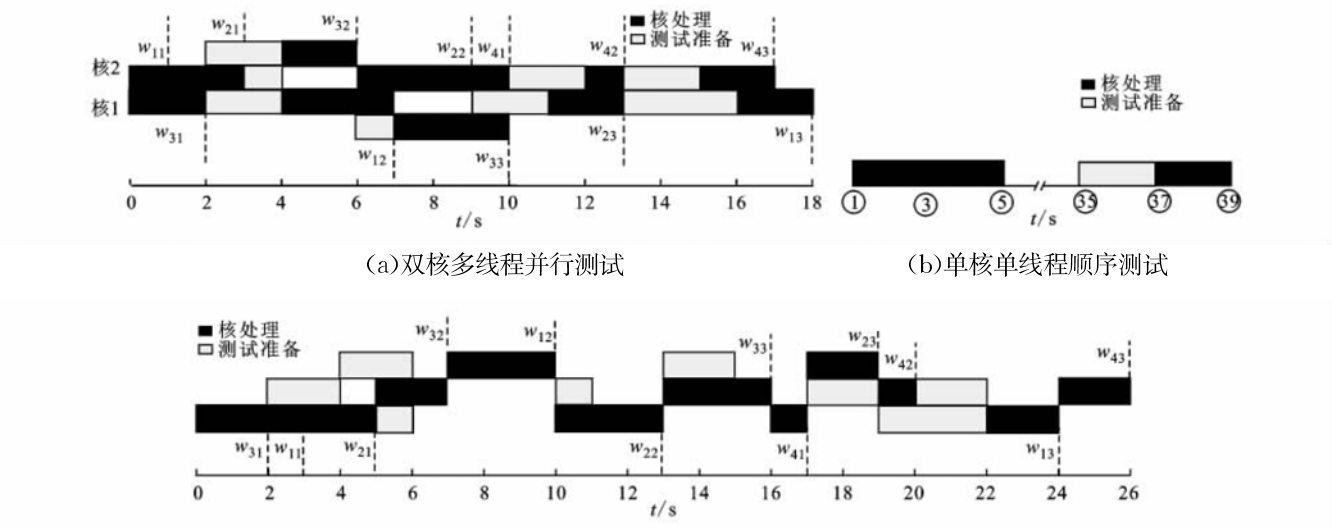
$$\boldsymbol{O}^{i\times j}=\begin{pmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\end{pmatrix}$$

根据图 3 所示策略对任务进行排序测试,直到测试完成. 在图 2 所示任务的基础上,使用不同平台和不同编程方法的测试样本对比如图 4 所示.

在本文实验中,基于工作量的多核并行测试较之单核并行测试的效率提高了 30%,较之单核顺序测试的效率提高了 50%. 为了便于说明问题,本文实验的测试过程全程占用了处理器核,排除此限制后的多线程并行测试效率还有等于进一步提高. 但是,本实验的目的是验证、对比多核平台和单核平台的测试效率,在同样使用多线程技术进行编程的情况下,本文结果是可信的.

4 结束语

本文将多核技术应用于并行测试领域,并给出



(a)双核多线程并行测试 (b)单核单线程顺序测试

(c)单核多线程并行测试

图 4 使用不同平台和不同编程方法的测试样本对比

了多核测试平台软件测试结构，详细描述了多核并行测试平台构建需要解决的任务分解和任务调度问题.为此，又提出了任务、数据、数据流3种任务分解方式.同时，针对测试任务调度问题综合考虑了测试资源、计算资源和测试任务优先权等因素，提出了基于工作量的并行任务调度策略，优化了任务调度过程.最后，用实验证明了多核平台的测试效率.

参考文献：

[1] ZHU Xiaoping, XIAO Mingqing. Modeling on parallel test system based on object-oriented [C]// Proceedings of Instrumentation and Measurement Technology Conference. Piscataway, NJ, USA: IEEE, 2005: 2076-2081.

[2] KALOGEROPULOS S, RAJAGOPALAN M, TIRUMALAI P. Processor aware anticipatory prefetching in loops [C]// Proceedings of the 10th International Conference on High-Performance Computer Architecture. Los Alamitos, CA, USA: IEEE Computer Society, 2004: 106-117.

[3] AKHTER S, ROBERTS J. Multi-core programming [M]. Hillsboro, OH, USA: Inter Press Business Unit, 2004: 9-11.

[4] ADL-TABATABAI A R, HUDSON R L, SUBRAMONEY S. Prefetch injection based on hardware monitoring and object metadata [C]// Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation. New York, USA: ACM, 2004: 267-276.

[5] CHOI S, KOHOUT N, YEUNG D. A general framework for prefetch scheduling in linked data structures and its application to multi-chain prefetching [J]. ACM Transactions on Computer Systems, 2004, 22(2): 214-280.

[6] KIM D, SHEN J P, GIRKAR M. Physical experimentation with prefetching helper threads on Intel's hyper-threaded processors [C]// Proceedings of the International Symposium on Code Generation and Optimization. Los Alamitos, CA, USA: IEEE Computer Society, 2004: 27-38.

[7] KUSISK A. Decomposition of the design process [J]. ASME Transactions: Journal of Mechanical Design, 1993, 115(4): 687-695.

[8] MAHESWARAN M, SIEGEL H J. A dynamic matching and scheduling algorithm for heterogeneous computing systems [C]// Proceedings of 7th Heterogeneous Computing Workshop. Los Alamitos, CA, USA: IEEE Computer Society, 1998: 57-69.

[9] ALHUSAINI A H, PRASANNA V K, RAGHAVENDRA C S. A framework for mapping with resource co-allocation in heterogeneous computing systems [C]// Proceedings of Heterogeneous Computing Workshop. Los Alamitos, CA, USA: IEEE Computer Society, 2000: 273-286.

[10] PARKHURST J, DARRINGER J, GRUNDMANN B. From single core to multi-core preparing for a new exponential [C]// The International Conference on Computer Aided Design. San Jose, CA, USA: IEEE CASS/CANDE, 2006: 67-72.

[11] ALAM S R, BARRETT R F, KUEHN J A, et al. Characterization of scientific workloads on systems with multi-core processors [C]// IEEE International Symposium on Workload Characterization. Los Alamitos, CA, USA: IEEE Computer Society, 2006: 225-235.

（编辑　苗凌）

一种新型高阶两通道时间交织 ΣΔ 调制器系统结构

杨骁，陈贵灿，程军，徐晓云

（西安交通大学电子与信息工程学院，710049，西安）

为了减小两通道时间交织 ΣΔ 调制器中通道之间系数失配引起的折叠噪声，提出了一种新型的高阶两通道时间交织 ΣΔ 调制器的系统结构.通过在传统噪声传递函数(NTF)中增加一个 $z=-1$ 的零点，得到了一种新的 NTF.新增的零点减小了 NTF 在高频处的幅值，从而能够减小两通道时间交织调制器结构中由于系数失配引起的折叠噪声.以实现新 NTF 的单通道单环 4 阶 3 b 前馈分布型 ΣΔ 调制器结构为原型，利用多抽样率系统和块数字滤波器基本原理，得到其对应的两通道时间交织系统结构.在两个通道的系数存在1%的失配条件下，调制器的信号噪声失真比只降低了 3.1 dB，这表明系数失配对该调制器的性能影响很小.