程序调试、监测与性能分析的一体化工具*

刘 强 张兆庆 乔如良

(中国科学院计算技术研究所高性能中心 北京 100080)

E-mail: qliu@bbti.com.cn

摘要 随着高性能芯片及高性能计算的应用,人们对调试器的要求不再仅局限于"正确性调试",而是更进一步要求进行"性能调试",即在正确性的基础上,通过对程序的细调而获得最佳性能.正确性调试和性能调试通常由独立的工具支持,这种模式已不能很好地支持高性能计算.介绍了一个为高性能SIMD芯片设计的同时具有正确性调试和性能调试功能的可视化工具.它集程序调试、行为监测和性能分析为一体,改善了高性能程序的开发周期,为高性能计算提供了强有力的支持.

关键词 调试,性能分析,行为监测,事件采集,断点,程序插桩.

中图法分类号 TP311

程序调试是程序开发中最基本的活动之一.当算法设计好之后,程序员往往进入编辑→编译→调试的开发周期.几乎所有的开发环境中均有功能完备的源码级调试器的支持.另一方面,随着高性能计算日益走向成熟,对高性能计算中性能分析工具的研制则成为近年来的一个研究热点.这是显而易见的:若高性能计算没有性能分析作为基础是不可想像的.性能分析通常采用了基于事件的方法,即静态时对源程序或运行库插桩,运行时采集感兴趣的事件,运行后根据事件对程序进行性能分析.当性能分析结果不满意时,则进一步修改程序,再次进入编辑→编译→调试周期.基于以上调试和性能分析的模型,高性能计算领域中程序的开发周期如图1所示.

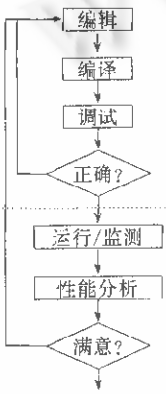

图1 具有性能分析的开发周期

与传统的开发周期相比,性能分析已成为高性能开发环境中重要的一部分.目前流行的高性能计算平台,如PVM,EXPRESS均支持以上开发周期.笔者在我国大规模并行计算机曙光1000的研制中,为曙光1000研制了性能可视化工具ParaVision,其机制与上述类似.然而在对ParaVision的试用过程中却暴露了如图1所示的开发周期存在的问题:(1) 该开发周期将正确性调试和性能调试割裂为两个阶段、两个工具,而实际开发过程中以上两个阶段往往是并发、交互进行的.(2) 性能分析在程序执行之后进行,缺乏交互性、实时性,尤其对运行时间长的程序,开发效率低,开发周期长.

以上开发周期的不足之处使我们认识到将程序调试和性能分析结合在一起的重要性.在为一个高性能SIMD芯片研制开发环境的项目中,我们将以上想法付诸实现,其结果是一个集成了程序调试、行为监测和性能分析为一体化的工具——Visual-IDP(visualized integrated debugger and performance analysis tool,简称IDP).本文介绍Visual-IDP的设计实现以及它对高性能程序开发周期的全新支持.

1 目标机结构和开发环境

目标机是高性能的单指令流、多数据流(SIMD)处理器,其结构如图2所示.

* 本文研究得到国家863高科技项目基金资助.作者刘强,1959年生,博士,主要研究领域为编译系统及环境工具.张兆庆,女,1938年生,研究员,博士生导师,主要研究领域为编译系统及环境工具.乔如良,1937年生,研究员,主要研究领域为编译系统.

本文通讯联系人:刘强,北京 100076,北京市海淀区知春路76号翠宫饭店BBT公司

本文1997-11-13收到原稿,1998-04-01收到修改稿

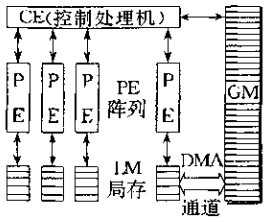

图2　目标机体系结构

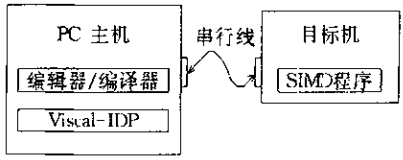

图3　主机-目标机模型

从体系结构的角度看,目标机由控制处理机 CE,处理单元阵列 PE,局部存储器 LM,全局存储器 GM 和高速 DMA 通道五部分组成.从性能的角度考虑,为使处理机处于高性能运转状态,应着重于以下两点:

(1) PE 阵列计算.因为 PE 阵列是用于计算的主要动力,因此,使 PE 阵列始终处于忙状态有利于达到最佳性能.PE 阵列的计算需要等待数据从 GM 传输到 LM.

(2) LM-GM 间的 DMA 通信.应尽量减少通信开销.

对目标机的开发采用了主机-目标机模型,如图 3 所示.主机为 PC 工作站,主机和目标机之间通过串行口进行通信联系.用户在 PC 机上编写程序,用交叉编译器生成目标机的执行代码,通过串行口将目标码装载至目标机 GM 中而执行程序.Visual-IDP 通过串行口与目标机通信,实现对 SIMD 程序的远程调试和监测.

2　IDP 的结构和实现

IDP 集成了调试器、程序监测和性能分析的功能,结构相对复杂.我们在设计中充分利用了面向对象的原则,注重模块化、层次化和封装性.图 4 给出了 IDP 的总体结构.

2.1　Client/Server 模型

开发环境的主机-目标机模型导致了 IDP 的 Client/Server 结构.Client 端运行在主机上,Server 端运行在目标机上,二者通过串行进行通信交互.

IDP Client:Client 接受用户输入,通过串行口与目标机通信,以完成特定的调试、监测任务.IDP Client 在结构上可以分为以下 4 个层次的功能模块:① GUI:图形用户接口,并以可视化的形式显示程序行为和性能数据.② 功能层:IDP 的主体,由调试器、监测器和性能分析器组成.它由 GUI 驱动,并将其功能最终分解为对 IDP Client API 的调用.③ IDP Client API:IDP 客户端应用程序接口.Client API 是对目标机的封装和抽象,其具体实现是通过对目标机发出请求而完成.Client API 封装了远程调试的细节,功能层可以不考虑是在本地调试还是在远程调试,这对整个系统的模块化有重要的贡献.Client API 是相对比较原始的操作,主要功能包括:目标程序控制和对目标机状态的存取.④ 通信层:对串口的封装,命令和数据的传递,采用简单的发送-应答协议.通信层的设计封闭了所有串口操作的细节.

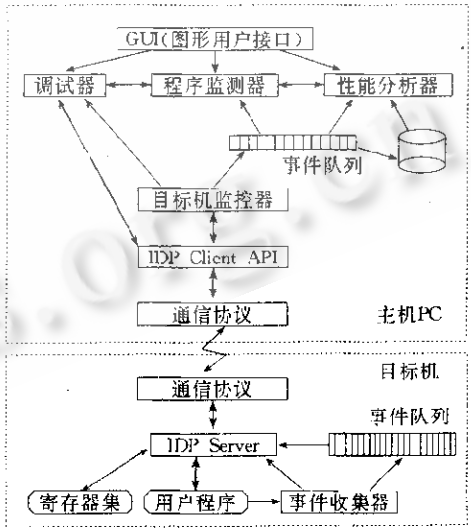

图4　IDP的总体结构

IDP Server:对目标机的一切操作集中在 Client API 上,由于主机上无法直接对目标机操作,因此,Client API 的功能由目标机上的 IDP Server 代理实现.IDP Server 等待来自串口的服务请求,通过对目标机寄存器和内存的直接操作而对外来请求提供服务.

2.2　目标机监控器

目标机监控器在 IDP 中起着重要的作用.它监测目标机程序的运行状态,同时为调试器和程序监测器提供

相应的信息.目标机监控器主要监控以下两类事件:① 程序调试事件.它是由断点命中、单步运行等调试功能引起的目标程序运行状态的改变.目标机监控器将该类事件发送给调试器,调试器接受到事件后作出相应的调试动作.② 程序行为事件.与调试事件不同,该类事件是由事件收集器发出的与程序行为和性能有关的事件.目标机监控器收到该类事件后,将事件放入主机中的事件队列并进一步通知程序行为监测器.程序行为监测器读取事件队列,可视化程序的行为.

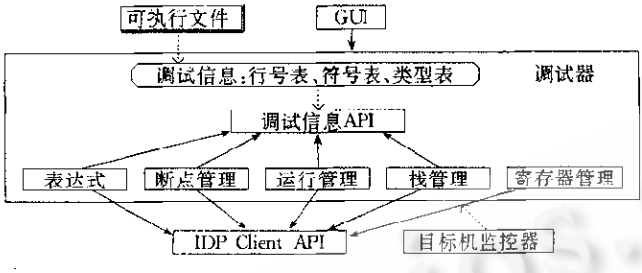

图5 调试器结构及和相邻模块的关系

2.3 调试器

图5显示了调试器的组成结构以及它与相邻模块的关系.调试器包括3个层次的逻辑模块:调试信息、调试信息API和调试器核心功能.一方面,调试器接受GUI的命令,通过对Client API的调用而完成核心功能的实现.另一方面,调试器接受目标机监控器的消息而获得目标程序的状态.鉴于调试器设计已经是一门成熟的技术,我们这里重点解释远程调试的实现.

与调试本地机程序不同,IDP需要处理远程调试.调试功能的实现是由Client端和Server端共同协作完成.我们以对断点的处理为例:

```
a—b    →    ld r0,&a    →    * DebugTrap
            st &b,r0          st &b,r0
```

在上例中,设置断点时,我们将断点处指令改为DebugTrap.DebugTrap是一条非法指令,当程序运行到断点而执行DebugTrap时,控制处理器CE产生非法指令例外.非法指令例外处理程序通知IDP Server.在主机上,目标机监控器通过与IDP Server的交互而检测到该例外的发生.目标机监控器将该事件通知调试器而使调试器获得控制,以作进一步的处理.

2.4 程序监测

相对于调试器而言,程序监测是IDP的另一个重要方面.程序监测一方面用于可视化程序行为,一方面为性能分析器提供数据.程序监测和性能分析的基础在于程序事件[11],围绕着程序事件,我们要解决事件类型的划分、事件产生、事件检测、事件传递、事件显示和事件分析各方面的问题.

2.4.1 事件类型

参考第1节描述的目标机的性能特点,IDP应监测对程序的行为、性能有影响的事件.

表1 IDP所监测的几种主要事件类型

事件类型	描 述	主要数据	事件范畴
FUNC_ENTRY	进入一个函数体	时戳,函数编码	结构事件
FUNC_EXIT	退出一个函数体	时戳,函数编码	结构事件
SIMD_START	PE计算开始	时戳	计算事件
SIMD_END	PE计算结束	时戳	计算事件
WAIT_START	目标机开始等待通信	时戳	通信事件
WAIT_END	目标机结束等待通信	时戳	通信事件
DMA_START	DMA开始运行	时戳,通信方向	通信事件
DMA_END	DMA结束运行	时戳	通信事件

表1中的事件分为以下几个范畴:

(1) 程序结构事件:监测程序的流程和位置,为性能数据提供源程序所对应的信息.

(2) 程序计算事件:监测程序的串行计算和并行计算,用于统计程序的并行性.

(3) 程序通信事件:监测DMA的通信开销,如DMA_START.

以上事件与程序的行为、性能紧密相关.对上述事件的分析,程序员能够跟踪程序热点,发现程序性能瓶颈.

2.4.2　事件的产生、检测和传递

事件的产生、检测和传递机制三者之间紧密联系而成为一个整体.在图4的IDP总体结构中,自上而下,它由目标程序、事件检测器、IDP Server、IDP Client API和目标机监测器协同完成.

对事件的产生机制,我们采用了对用户程序和运行库插桩(Instrumentation)的方法.有更加精致的方法跟踪程序事件,例如,采用总线窃听[2,3].这种技术的突出优点在于有利于减少对用户程序的干扰,但需要硬件支持且获得的数据过于原始,不利于抽象.由于我们受到硬件的限制而采用插桩方法,这对用户程序有一定影响,但实践证明,此方法行之有效.简言之,我们通过在用户程序和运行库中插入模块GenEvent(EventType,EventData)而实现事件产生.GenEvent在实现上为宏,以达到高效率.例如,当DMA开始通信时,以下代码产生该事件:

```
Q→Type=DMA_START;
Q→Time=Clock;
Q++;
If(Length(Q)>MAX_EVENT) NotifyIDPClient();
```

在以上代码中,Q是目标机中的事件队列.我们在全局内存GM中保留了一块内存区域用来存放事件队列.MAX_*EVENT*是队列中能够容纳事件的最大个数.考察以上代码的执行效率:以上代码翻译成汇编语言只有5条机器指令,而对于函数调用NotifyIDPClient,If语句的条件决定了每产生MAX_*EVENT*条事件才调用一次该函数,当MAX_*EVENT*较大时,产生每条事件的开销会非常小(几个机器周期),从而对用户程序的影响也非常小.

进一步考察事件传递.当事件队列满时,事件信息需要从目标机传送给主机.这由Client端和Server端配合完成.在目标机上,事件队列满时执行函数调用NotifyIDPClient,该函数机制如下:

```
NotifyIDPClient(){
    EventFlag=1;
    DebugTrap;
}
```

NotifyIDPClient只有两条指令.DebugTrap指令使用户程序导致异常而暂停,这与断点机制一样.不同的是,NotifyIDPClient在自我暂停之前设置EventFlag标志值,在主机上,目标机监测器根据该值而区分程序行为事件和调试事件.图6给出了事件传递、调试和目标机监控器三者之间的交互关系.在图6中,调试断点和事件传递的DebugTrap指令均使目标机产生例外而停止.目标机监测器根据EventFlag的值而判断目标机程序停止的原因.当判断程序停止是由事件传递产生时,则将事件信息从目标机队列读取到主机的事件队列中,并复位EventFlag,之后使目标程序继续运行.

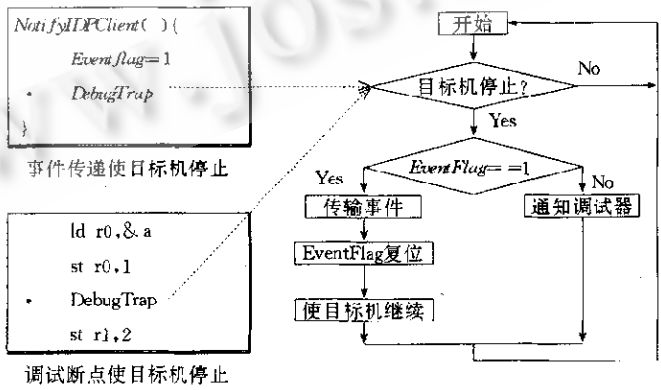

图6　事件传输、调试和目标机监控器三者之间的关系

以上介绍了事件的产生和传递机制.目标机和主机之间的精巧配合,使我们以很低的开销和复杂性实现了这一任务.

2.4.3 程序监测器

程序监测器以两种方式工作:后台方式和可视化方式.在后台方式下,程序监测器简单地将事件队列转储到磁盘文件中以用于事后分析.可视化方式下,程序监测器将程序的运行状态以图形可视的方式显示给用户.

2.5 性能分析器

性能分析器通过对事件序列的分析而获得性能数据.有事件采集作为坚实的基础,性能分析器的设计就相对容易了.文献中有大量的基于事件的性能分析方法,这里不再深入.简言之,IDP分析以下性能数据:(1) 目标程序的并行性以及目标程序的有效计算时间.(2) 目标程序计算与DMA通信之间的并行以及DMA通信的通信情况.

3 IDP所支持的开发周期

第2节中给出的IDP设计和功能支持图7所示的开发周期.与图1所示的开发周期相比,IDP所支持的开发周期具有以下特点:① 调试周期与性能分析周期合成为一个,而不是两个独立的部分.② IDP支持用户同时进行程序调试、程序行为监测和性能分析.

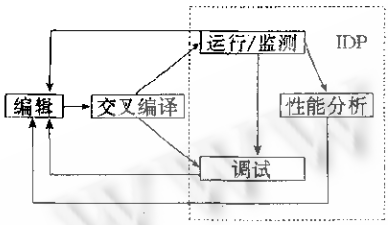

图7 IDP所支持的开发周期

图7所示的开发周期对传统的开发周期的改进正是IDP设计的初衷.从用户使用的角度来看,用户可以将IDP看作一个调试器,在调试时,可以监测程序行为、观察程序性能;另一方面,用户也可以将IDP看作一个程序性能行为监测器.在监测程序时,可以利用调试器的强大功能,如看值、看栈和单步等.IDP的这种灵活性有力地支持了用户程序的开发.

4 结束语

IDP是近年来我们从事高性能计算开发环境研究工作中的成果.IDP集成了程序调试、程序行为监测及性能分析功能,有效地支持了高性能程序的开发.高性能计算需要良好的开发环境的支持.IDP是我们在这方面的探索.IDP的想法和设计原则可适用于MPP(大规模并行处理)系统,从而此技术可推广到更为广阔的领域.

参考文献

1 Worley Patrick H. A New PICL Trace File Format. Oak Ridge National Laboratory: Mathematical Sciences Section, 1992

2 TSAI Jeffrey J P, MEMBER, IEEE *et al*. A noninterference monitoring and replay mechanism for real-time software testing and debugging. IEEE Transactions on Software Engineering, 1990,16(8)

3 Dodd Paul S, Racishankar Chinya V. Monitoring and debugging distributed real-time programs. Software-Practice and Experience, 1992,22(10)

An Integrated Tool for Debugging, Monitoring and Performance Analysis

LIU Qiang ZHANG Zhao-qing QIAO Ru-liang

(*High Performance Center Institute of Computing Technology The Chinese Academy of Sciences Beijing* 100080)

Abstract As the development of high-performance computing, the requirement for debugging is not only limited in "correctness debugging", but also for "performance debugging", which means to achieve the best performance through program tuning. The correctness debugging and the performance debugging are usually supported by the separate tools, which couldn't sufficiently support high-performance computing. A visualization tool is presented in this paper that supports both "correctness debugging" and "performance debugging" for a high performance SIMD chip. This tool incorporates the functionality of debugging, the behavior monitoring and the performance analyzing. It greatly improves the development cycle of high-performance computing programming.

Key words Debugging, performance analysis, behavior monitoring, events collecting, breakpoint, program instrumentation.