并行程序性能可视化分析系统的研究与实现

马　捷

指导教师：郑守淇　教授

西安交通大学计算机系

1998.07

并行程序性能可视化分析系统的研究与实现

摘 要

随着并行技术及并行应用的不断发展，并行程序的设计、开发、调试、优化成为并行技术应用的关键。本文首先介绍了一个性能可视化分析系统的框架模型，然后提出了一个多层次的可扩展的性能可视化模型。VENUS 系统是基于这一模型的一个性能可视化分析与优化的软件系统，它的可视化部分通过使用动态调整视图播放细节度等技术，使 VENUS 系统能够高效、准确地展现并行程序性能，提高了系统的可用性。本文详细论述了性能可视化模型的结构及 VENUS 系统的设计思想和实现技术，并在文章中给出了 VENUS 系统的一个应用实例，分析了 VENUS 系统的性能。

[关键词] 可视化 视图 并行性能 性能分析 并行虚拟机
[论文类型] 应用基础

Research and Implementation of a Parallel Program Visual Performance Analyzing System

Abstract

With the development of the parallel computing technique and application, parallel program designing, debugging and optimizing become a key technique of application. In this paper, a frame model of the visual performance analyzing system is introduced and a multi-level visualization model is discussed. VENUS system is a visual performance analyzing and optimizing system based on this model. By using dynamic-adjust-play-scale technique, VENUS system can visualize the performance of parallel program more efficiently. In this paper, the structure of the visual performance analyzing model, as well as the design and implementation techniques of VENUS system are discussed in detail. An example of application and performance of VENUS system are also included in the paper.

[Keyword] Visualization, View, Parallel Performance, Performance Analysis, PVM
[Thesis Type] Application

目 录

第一章 概 述

1.1 背景

随着并行技术及并行应用的不断发展，并行程序的设计、开发、调试、优化成为并行技术应用的关键。由于并行系统硬件体系结构、软件平台等方面的差异，并行程序在不同平台上的运行效率差别也很大。因此，并行程序的优化就成为并行应用方面的一个关键性问题。并行程序不同于以往的串行程序，它的执行过程较为复杂，具有一定的随机性和不确定性，运行过程不可再现。因而，并行程序的性能优化过程比较困难。这就使得对并行程序的运行性能进行监测、可视化及分析具有了非常重大的意义[1]。

为了充分发挥并行计算机的潜力，提高并行程序的性能，国外开展了大量的关于并行程序开发支撑环境的研究工作，其中并行程序的性能分析是重要研究之一。并行程序的性能分析大致采用三种方法：基于模型的性能分析方法；基于测量的性能分析方法；基于模拟的性能分析方法。可视化是近年来在性能数据处理分析中经常采用的技术。将数据以可视化的方式表现出来，对于了解数据变化的趋势、直观表现数据的内在含义有着十分重要的作用。该技术给用户提供直观明确的界面，有助于对系统性能的分析，定位系统瓶颈，因此得到越来越多的重视。

国内在并行机研究的同时也开展了并行程序开发支撑环境的研究，如清华大学在所研制的可扩展机群上开发了并行程序调试工具和并行程序行为监测工具。但其性能数据的表示过于简单，没有一个直观的可视化界面；同时，也不能对并行程序性能进行细致地分析，和给出关于克服性能瓶颈的具体的提示性信息。

1.2 国内外相关研究

1.2.1 ParaGraph

ParaGraph 是用于显示并行应用程序的实际运行行为和性能的软件，它提供一个可视的重放机制来重放真实并行应用程序在实际并行平台上的运行事件记录。目前，ParaGraph 主要提供基于事件跟踪文件的事后重放机制。通过分析并行应用程序运行过程中的性能数据记录，产生多个相关的性能数据可视化视图，从而可以从不同的角度分析并行应用程序的性能，使性能优化过程更加简单、直观。

ParaGraph 以事件跟踪文件作为输入。事件跟踪文件是由 Portable Instrumented Communication Library 产生，因此，ParaGraph 不需要特别的修改被跟踪对象的源代码，就可以自动地获取其性能信息。

ParaGraph 的主要特点还有，它提供了多个视图同时播放，相互对照的功能；其视图显示可以映射到各种不同体系结构的计算机；以及其易于移植和扩展的特性[2]。

1.2.2 Paradyn

Paradyn 是对大规模并行应用程序进行性能分析的一个辅助软件工具包。它通过使用动态指令插入技术和自动瓶颈测定技术，在线式地监测和分析并行应用程序的性能。Paradyn 由一个灵活的数据采集机制、一个瓶颈自动探测器和一个可视化用户界面组成，能够对运行时间较长的应用程序进行性能分析。Paradyn 近期也被用于 PVM 应用程序的性能分析。程序员在 Paradyn 环境中运行未修改的 PVM 应用程序，Paradyn 会在程序的运行过程中自动插入和修改其指令，系统地探测造成性能问题的原因。在很多情况下，Paradyn 可以分析出造成性能问题的主要原因，并指出对应的程序代码。

Paradyn 采用了 W^3 搜索模型，使用户(程序员)能在系统中的决策支持部分的协助下，以动态在线式(on-the-fly)的数据采集方法采集性能数据。W^3 搜索模型基于回答下述三个独立的问题：为什么并行应用程序的性能较差(Why)，性能瓶颈在什么位置(Where)，和什么时间发生的性能问题(When)。通过对这三个问题的不断分析，系统可以指出应用程序的性能问题形成的原因。

Paradyn 能够运行于 CM-5(Thinking Machine)，SUN 工作站及 PVM 环境。它可以在这些平台上测量同构的程序[3]。

1.2.3 MUCH 系统

MUCH(MUltiprocessor Class Hierarchy)是 Johannes Kepler 大学研究的一种并行程序可视化类库，它用于帮助用户建立不同多处理机系统结构的可视化模型。用户可以利用 MUCH 所提供的类(结点类、互连网络类等)和一些相应的方法来建立不同系统结构多处理机的一种可视化工具。利用 MUCH 与其它并行程序可视化及分析系统配合，可以更有效地对并行程序进行调试及分析优化[4]。

1.2.4 IPCE

IPCE 是清华大学研制的可视化人机交互集成开发环境。它是在对 XPVM 和 PVM 修改的基础上集成的一个综合性的并行程序开发环境。它为应用程序员和最终用户提供了编辑、编译应用程序，配置、监视并行环境，加载、监控并行任务以及并行任务的性能评测和并行调试等功能[5]。

1.2.5 PVM 监控调试工具

- **XPVM**

XPVM 是一个图形界面的 PVM 控制台，它提供了一个有效的方式来查询和设置虚拟机器及各个主机的状态；同时 XPVM 还提供了性能监视和函数调用级调试的图形界面。在性能监视和函数调用级调试过程中，XPVM 利用了 PVM 内部的性能监测机制采集数据，采用了 SDDF(Self-Describing Data Format)格式的性能数据文件进行数据交换。

- **PGPVM**

与 XPVM 类似，它提供一个 PVM 的跟踪机制，用于产生标准 ParaGraph 的跟踪文件，该文件可以用于 ParaGraph 可视化工具。

- **PVaniM 2.0**

PVaniM 提供了在线或重现机制下的可视化分析工具，一方面，它利用一些技术使得

系统在监视 PVM 运行情况时的开销最小，另一方面，也同时提供了多种可视化的视图[6]。

1.2.6 ParaVision

ParaVision 是曙光-1000 上的并行程序可视化工具。它在逻辑上分为四个部分：事件检测器、事件收集器、行为监控器和性能分析器。事件检测器同目标程序一起运行于每个结点机上，通过 Nx 通信函数将收集到的事件传给事件收集器。ParaVision 对事件的检测采用了软件的方法，它通过编译程序处理结点程序的内部事件，通过改造库函数处理结点程序间的交互事件，使得对事件的检测完全透明于用户。事件收集器运行于主机上，取代了标准的加载程序的功能，负责管理程序的加载、启动、结束，同时，收集并处理事件检测器送来的事件，送给行为监控器或存储到磁盘以备后用。行为监控器和性能分析器处理所得到的事件，可视化程序行为和性能。

ParaVision 对程序行为进行了可视化表示，通过动态执行路径和程序状态的显示使程序员能够了解程序动态的执行过程及程序的状态；针对曙光-1000 的特点，着重对处理机间通信进行监测，不仅对结点间的交互通信给出了直观的表现，而且使程序员能够很容易地发现处理机之间的死锁。ParaVision 的性能分析部分通过对收集到的事件进行统计分析，对应用程序提供了多种性能参数，其中包括每个处理机的计算时间、通信开销、通信量、各处理机的利用率以及整个系统的负载平衡情况和并行性[7]。

1.2.7 EDPEPPS 仿真系统

EDPEPPS(Environment for Design and Performance Evaluation of Portable Parallel Software)是 Westminster 大学的研究项目。EDPEPPS 是基于快速原型机制进行并行程序设计的一个工具，它使用一种面向仿真的语言进行程序原型的设计，并在特定的仿真平台上进行仿真，最终将所获得的性能数据以可视化的方式表现出来。其面向仿真的语言提供了与 PVM 相同的并行机制[8]。

1.3 论文研究的内容

本文是结合 863 项目**大规模并行处理体系结构的关键技术**，针对基于消息传递机制的并行系统性能进行可视化及分析的研究。除了基于 863 项目开发了 VENUS 系统(Visual Environment of Neocomputer Utility System)的可视化、分析及系统界面程序外，还对性能数据可视化进行了较深入的研究，从理论上建立可视化的基本模型，并在实现 VENUS 系统的过程中，对该模型的性能进行了测试分析及改进，在理论和实践上对并行系统性能可视化进行了研究。

本文首先在第二章中提出了一个对并行应用程序进行性能可视化分析的系统的框架模型，并对其构成中的每一部分进行了详细的分析；然后，在第三章中着重阐述了性能可视化模型，详细论述了该模型的分层结构及每一层的功能；第四、五、六章分别介绍了 VENUS 系统的详细设计与实现，使用 VENUS 系统进行性能分析优化的实例，并分析了 VENUS 系统性能；最后，论文在第七章中总结全文，并对论文中所建立的性能可视化模型以及下一步的研究工作进行了分析和讨论。

在第二章中提出的性能的可视化分析系统是一个框架式的模型，它将可视化分析系统分解成三个相对独立的模块——性能监测、性能可视化与性能分析。这三个模块中，性能

监测是与系统硬件关系最为密切的一个模块，不同的硬件需要用不同的方式进行监测。本文中给出了几种常见的性能监测方式。性能分析是对所监测的性能(数据)进行分析的一个模块，它虽然与硬件的关系不十分密切，但不同的分析方法在不同的硬件平台上的分析效果不同，它对硬件平台也有一定的依赖性。在这一章中对这两个模块只提出了一个框架式的模型，具体的实现还需要根据具体的硬件平台进行分析。

性能可视化模块是用可视的方式将性能数据或性能分析结果表示出来的功能模块，它是性能可视化分析系统中与硬件的关系最为简单的模块。在本文第三章中，对性能可视化进行了更进一步的分析和研究，提出了一个与系统硬件平台无关的性能可视化模型，并使其在实现过程中硬件对它的影响降至最低程度。

VENUS 系统是性能可视化分析框架模型的一个实例，它具有一套完整的可视化体系结构，同时又从实用的角度上在设计中对它进行了优化，为用户(程序员)提供了一个易学易用的集成环境。本文在第四章中讨论了 VENUS 系统的设计与实现，在第五章中以 EP 问题为例介绍了如何利用 VENUS 系统进行并行应用程序的分析优化工作，并在第六章中分析了 VENUS 系统的几个应用实例，对 VENUS 系统对并行应用程序的优化效果进行了分析。

第二章 性能可视化分析系统的框架模型

2.1 概述

性能可视化分析系统框架模型是一个与硬件平台无关的软件系统的框架模型。由于它与硬件平台无关，这就要求这个模型具有良好的可移植性。在这里，可移植性有两方面的含义：一方面是这个模型从实现的角度上不依赖于特定的硬件平台，另一方面是指这个模型从性能的角度上也与硬件无关，也就是说，从性能分析功能的角度上，这个模型应能适用于多种体系结构的硬件平台。

并行计算机从开始研究起至今，已产生了许多不同的体系结构，从流水线、向量机到多处理机、多计算机。但任何一种系统结构都有其局限性，每种系统都有其性能特点。例如：对于多计算机体系结构上应用程序因处理机间通信过多而造成的性能下降，在多处理机体系结构上可能并不存在。因此，性能可视化的框架模型除了需要考虑实现的方便外，还要考虑到不同体系结构平台在性能上的特殊性，为有效地实现不同体系结构上的性能分析提供一个合理的框架结构。

在本章的下一节中，将着重从总体上建立起性能可视化分析系统的框架模型，并在第三节中对构成这一框架模型的各个模块进行分析。

2.2 系统的总体框架

2.2.1 系统的总体框架

性能可视化分析系统是对并行程序进行性能分析，并以可视化方式表示出来的一个辅助开发及优化系统。并行程序的性能分析大致采用三种方法：基于模型的性能分析方法；基于测量的性能分析方法；基于模拟的性能分析方法。由于基于模型或基于模拟的性能分析方法都是针对特定平台给出特定的模型进行分析，而这里所要建立的是一个与平台无关的框架模型，因此，在这个框架模型中，我们采用了基于测量的性能分析方法。它不依赖于对硬件平台所建立的模型，直接根据实际测量所得的数据(性能数据集)进行分析，从而摆脱了分析系统对硬件平台的依赖性。

性能可视化分析系统从功能上可以分为四个模块：性能监测模块、性能可视化模块、性能分析模块和用户界面及辅助管理模块。

如图 1所示，性能可视化分析系统的框架模型是以数据(性能数据集)为中心的一个框架模型，可视化分析部分通过对性能数据集的访问来获得应用程序的性能状况，并由此进行分析与可视化表示。同时，可视化模块与分析模块对性能数据集进行处理后，产生一定的综合结果，为了区别它们与原始的性能数据，这个框架模型中不把这一部分数据放入性能数据集，而是在各个模块之间依照性能数据交换协议进行交流。这样，分析系统的主观因素不会影响到性能监测模块对应用程序性能的客观监测结果。

性能监测模块是系统中最基本的模块，整个系统通过它来获取应用程序在硬件平台上的性能，并以性能数据集的形式表示出来。一个应用程序的运行性能可以通过记录该程序的运行时间、同步、通信及其它一些重要事件发生的时间(时刻)和相关信息来表示。我们可以通过对上述数据(性能数据集)的分析，重构出其运行状况。同时，针对特定的需求，可以对应用程序进行特殊的监测，从而描述出其某一侧面的性能状况。

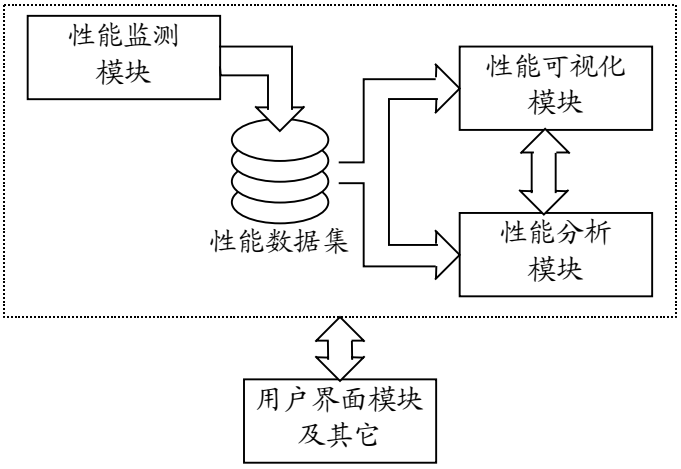


图1 性能可视化分析系统模型的功能框图

性能可视化模块是系统中的一个主要模块。简单地说，它的功能就是把系统内部的数据以直观的方式表示出来。首先，它需要对性能数据集中的数据进行分析理解，重构出应用程序运行的性能状况；然后，根据用户所需，以不同的形式表示出应用程序性能侧面。另一方面，系统中性能分析模块所得出的优化结论也应在这里表示出来。

性能分析模块是系统中另一个主要模块。它分析系统内部所获取的性能数据集中的相关性能数据，根据已有的经验(模型)以及系统内部的逻辑模型，通过推理产生出相应的分析结果。由于没有一个模型能够适用于所有的硬件平台，这里所提出的经验模型实际上是一种基于测量的模型。它是通过预先对硬件进行评测，并按照标准评测程序所得的结果来建立机器的模型。在对应用程序进行性能分析时，评价的标准与待评的性能数据都是通过实际测量而得来的，其评价结果具有一定的合理性。

用户界面模块及其它辅助功能模块为系统提供一个友好的可视界面和一些管理功能。用户界面的好坏是关系到一个应用系统是否能够被有效地使用的关键问题之一。随着计算机软件业的不断发展，用户界面的风格也在不断的变化，在本章所述的框架模型中，没有对用户界面的实现做过多的限制，只是从总体上提出了一些设计原则，作为系统在实现过程中的规范。此外，为了协调各模块的工作，性能可视化分析系统还应对可视化过程进行一定的管理。

2.2.2 模块式(插件式)系统结构

性能可视化分析系统框架模型采用模块式(插件式)的系统结构。在上一节中，本文分析了性能可视化分析系统的框架模型，并介绍了该框架模型的四个组成模块。由于这四个

模块的功能较为独立，因此，在这个框架模型中提出了这种插件式的系统结构。插件式系统结构一方面能够简化系统的实现过程，另一方面提高了系统的可用性及可重用性。系统结构如图 2所示：

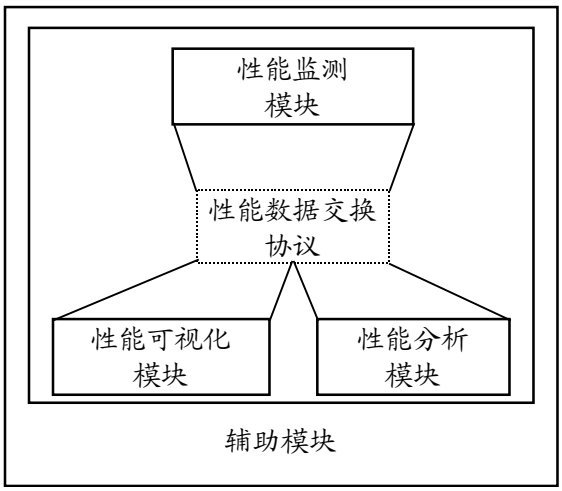


图2 性能可视化分析系统模型的模块结构

在系统的模块结构图中，四个模块分别对应系统框架模型中的四个功能模块。性能监测、性能可视化及性能分析模块由性能数据交换协议联系起来，通过性能数据交换协议进行模块间的信息交换。辅助模块在最外层，将系统中其它三个模块对用户(程序员)屏蔽，为用户(程序员)构造一个良好的使用环境。

在系统模块结构图中，性能数据交换协议处于中心位置。实际上，性能数据协议是性能监测、性能可视化及性能分析模块的接口标准，凡是满足性能交换协议要求的模块，均可以装配在系统之上。性能数据交换协议的主体是性能数据集规范，根据这个规范，性能监测模块对系统性能进行监测，并产生性能数据集；性能可视化及性能分析模块读取并分析性能数据集，并进行进一步的工作。这样，除了性能分析与性能可视化部分根据性能数据交换协议的其它部分进行直接地少量分析数据的交换外(不是必需的)，这三个模块的运行是不依赖于其它模块的。也就是说，这些功能模块一定程度上具有可独立运行的能力。

采用模块的方式来构造一个系统，对于每一模块分别进行设计编程，可以使得系统的研制得以并行进行；模块间的关系由明确的协议规定，各模块的相关性减小，工作得以简化；由于各模块统一设计协调，分别进行设计开发，使得模块分工协同效果优化，并防止错误在模块之间扩散蔓延，有利于系统的可靠性的提高。同时，由于性能可视化分析系统的各个部分在功能方面也相对独立，因而每个模块都可以作为一个实用程序单独使用，或作为其它系统中相应的功能模块，从而可以更灵活地对并行应用程序的性能进行分析优化，提高了系统各模块的可用性和可重用性。

构成性能可视化分析系统的四个模块的功能相对独立，对硬件的依赖程度也不尽相同。性能监测模块要求获取系统平台的相关性能、应用程序的相关性能等，对系统的硬件

平台依赖较大；性能分析及性能可视化模块只是根据前一部分提供的性能数据集及并行系统的体系结构的抽象模型进行分析与可视化，对系统的硬件平台依赖较小；用户界面及其它一些管理模块只与部分硬件(如显示设备等)有关，与并行体系结构无关。分模块进行设计实现，各个模块的可重用性都得到提高。

2.3 系统主要模块的设计

2.3.1 性能监测模块的设计

性能监测是并行应用程序性能分析的第一步，它通过采样、跟踪等方式将系统的性能数据记录下来，作为性能分析的依据。常用的性能监测方式有采样法和事件跟踪法两种。采样是按照一定的采样周期对所关心的系统性能数据进行记录；事件跟踪则是由所关心的系统事件触发跟踪机制，从而记录事件发生时(同时也是系统状态发生改变的时刻)的系统性能参数。

- **采样法**

采样是按照一定的采样周期对所关心的系统性能数据进行记录。在采样时，为了能够获取系统的性能变化，根据采样定理，采样频率至少是原数据变化频率的二倍。这样，一个运行时间较长的应用程序将产生大量的性能数据。采样法适用于由系统硬件实现，系统由硬件时钟以固定的周期对系统性能数据进行记录，不需要对应用程序代码(源代码或二进制代码)进行修改。此外，当采样频率足够大时，根据所采集到的性能数据可以精确地重构应用程序性能变化的状况。

- **事件跟踪法**

事件跟踪是由所关心的系统事件触发跟踪机制，从而记录事件发生时的系统性能参数。事件跟踪只记录了系统中重要事件发生的时刻及相关的其它参数，其数据量少于采样法所获取的数据量。但由于只记录了系统特定事件发生时刻的性能情况，事件跟踪机制对系统一般情况下(无特定事件发生时)的系统性能以及其它未考虑到的事件对系统性能的影响无法记录，从而在对系统性能描述的精度上低于采样法。另一方面，事件跟踪法由于所跟踪的事件类型较多，因而适于用软件实现。

- **事件跟踪的方式**

事件跟踪可以采用二进制代码插入、源代码修改以及更改系统链接库等多种技术实现。二进制代码插入是通过修改并行应用程序二进制代码的方式来加入事件跟踪代码。这种技术的优点在于：第一，它不需要并行应用程序的源代码；第二，它不需要重新编译，即用这种方法修改过的二进制代码仍是一个合法的应用程序，可直接运行。由于这两点的保证，二进制代码插入技术可使得对并行应用程序可以进行动态跟踪，即在跟踪的过程中可以动态调整跟踪策略，适当地增加或取消对某些事件的跟踪以获得最佳效果(事件记录最少，且记录的事件是最“有用”的事件)。但是，这种方法实现起来有一定的难度，它需要对机器指令有深入的了解。虽然有些机器及操作系统提供了动态改变应用程序执行代码的系统调用，但是，要达到动态地、智能化地对二进制代码进行分析与修改仍是一个较为困难的问题。在 Paradyn 系统中采用了这种技术。

源代码修改技术是一种直观的跟踪技术，实现较为方便。它通过在源代码中加入跟踪

语句(函数)来跟踪并行应用程序。同时，源代码修改技术还可以采用修改源代码中的特定代码(段)的方式，使得在对并行应用程序跟踪的过程中能够有效地监测应用程序特定代码(段)的执行情况，有利于从程序结构的角度对应用程序的性能进行分析。显然，源代码修改技术需要并行应用程序的源代码，并且在源代码修改后需要重新编译。同时，修改了源代码还可能造成程序语法错误甚至于改变了原来程序的语义功能。

更改系统链接库是通过对系统链接库的修改达到加入跟踪代码的一种技术。它通过用修改过的系统链接库(加入了跟踪代码)替换现有的系统链接库，使得并行应用程序在编译的过程中隐式地加入了跟踪代码。由于性能事件一般是以系统函数调用的方式表示的，因此采用更改系统链接库的技术基本能够满足对事件跟踪的要求。同源代码修改技术一样，更改系统链接库技术也需要并行应用程序的源代码并重新编译，此外，还需要系统链接库的源代码以便加入跟踪代码。虽然更改系统链接库技术也有对源代码的修改，但是由于这种修改只在系统实现的初期完成，不涉及使用过程中的源代码修改，同时，修改也仅限于更改系统链接库的源代码，因此，它的安全性高于源代码修改技术。

上述三种事件跟踪技术各有利弊，二进制代码插入技术实现困难，但能够跟踪无源代码的应用程序，适用范围广；源代码修改技术与更改系统链接库技术虽然适用范围较窄，但能够根据源代码信息更有效地对并行应用程序进行跟踪以获取更好的跟踪效果。在性能监测模块的实现过程中，可以同时采用多种跟踪技术，取长补短，获取最优效果。

2.3.2 性能可视化模块的设计

性能可视化是科学可视化的一种，它把并行应用程序的性能以可视的方式展现给用户(程序员)。可视化的性能可以是直接由性能监测模块所监测到的数据，或经由性能分析模块分析所得的结论，也可以是由可视化模块根据已有的数据进行统计分析所得的结论。严格地说，可视化部分所做的简单统计分析也应属于性能分析的范畴，但由于这部分工作过于简单，甚至可以在读取性能数据的同时进行，因此，在这里所提及的可视化也包括这一类的简单统计分析。

性能可视化首先要从性能数据集中抽取所要表现的性能数据，这一步称为**数据抽取**。数据抽取除了直接从性能数据集中筛选有效性能信息外，还包括对这些原始数据的简单加工(统计分析)和重组(为了显示的需要对数据进行排序组织)。经过数据抽取后，系统产生了待可视化的数据序列(性能数据子集)。

性能可视化的第二步是将性能数据子集映射到某种视图形式上，这一步称为**可视化表示**。对于同一个数据序列，可以采用不同的表示方式。例如，对于统计数据可以使用直方图、折线图、饼图或表格的形式；对于显示演变过程的数据，可以使用动画形式表示出每一时刻的状态，也可以采用以时间做为横轴的函数曲线来表示。可视化表示就是将第一步所获取的性能数据子集映射到某一可视化表示形式上。这样，根据不同的习惯和需要，就可以把相同的数据以更合适的方式表示出来。

经过了数据抽取和可视化表示，性能数据已经映射到某一可视化表示形式之上。性能可视化的第三步，**视图显示**，就是将可视数据真实地显示到物理设备(显示器)上。通过显示层处理，系统就将抽象的可视数据与实际物理设备联系起来。

性能可视化的三个步骤如图 3所示。以三个步骤完成可视化目的在于将性能数据分析

抽取方式与性能视图表示方式隔离，将性能视图表示方式与性能视图显示隔离。通过三层的隔离，使得这个模块更易于实现。下一章将专门讨论性能可视化，并把它抽象成为一个更完善的模型。

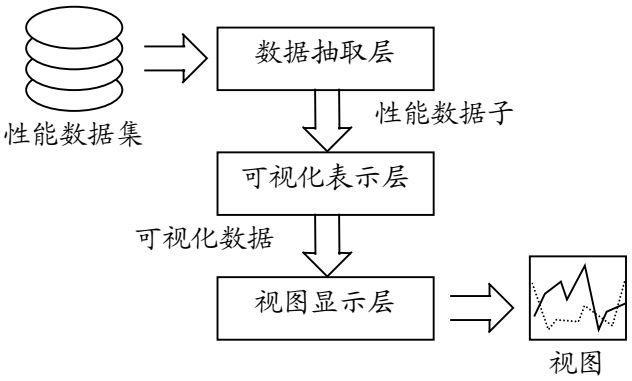


图3 性能可视化的三个步骤

2.3.3 性能分析模块的设计

性能分析模块是将已有的性能数据进行综合分析，以确定并行应用程序的性能状况的模块。涉及到程序的性能，必然与一定的硬件平台有关。为了能够更好地进行性能分析，这一模块采用了基于测量建立系统硬件模型，基于性能瓶颈假设进行验证分析的方式，同时在分析过程中采用可信度模型，进行模糊推理，提高能性能瓶颈的探测效果。

基于测量建立系统硬件模型是进行性能分析的第一步。由于硬件的不同，并行应用程序的执行效果也不尽相同。为了确保系统硬件不影响分析的结果，系统在分析并行应用程序之前首先利用一组基准测试程序(benchmark)对系统硬件的性能进行评测分析，建立起系统软、硬件的标准性能尺度，以便在其后的分析过程中对并行应用程序性能进行评测。基于测量建立系统硬件模型有两类实现的方式：一类是完全基于测量的方法，一类是基于测量分析的方法。完全基于测量的方法对系统软、硬件的所有性能尺度都进行实际测量，而基于测量分析的方法则是只测量基本性能尺度，对于其它的性能尺度以逻辑推导的方式按照一般性的软、硬件模型进行分析确定。这两种方法各有其优缺点：完全基于测量的方法能够不依赖于具体模型对软、硬件性能进行测量，但它所要测量的性能尺度过多，测量过程较为复杂。此外，要同时测量多个性能尺度，会造成基准测试程序间相互影响，以致加大测量误差。与之相反，基于测量分析的方法在基本性能尺度方面测量准确，程序实现简单，但对于所分析推导出的其它性能尺度则与软件模型有关。

基于性能瓶颈假设进行验证分析，是通过对比性能数据与性能尺度来验证性能瓶颈假设是否成立。在这里，我们采用可信度模型对并行应用程序是否含有性能问题进行模糊推理，从而避免了使用二值推理所造成的误差。

2.3.4 辅助模块的设计

辅助模块主要包括了可视化用户界面及系统设置、系统管理功能。用户界面根据需要

按照方便、易学、易用的原则进行设计。系统设置、系统管理功能则根据系统的需求进行设计。总之，辅助模块是系统核心与用户之间的中介，是用户与系统交流的媒体。这一部分的设计应以方便用户使用为主要目的。

2.3.5 模块间接口的设计

系统的核心由三个相对独立的模块组成。如图 2所示，这三个模块以性能数据交换协议为中心，通过这一协议进行数据交换。性能监测模块采集性能数据，形成性能数据集；性能可视化模块及性能分析模块读取性能数据集，并以可视化的方式表示或进行性能分析。性能数据集以文件(数据库)的方式存储，三个模块共享数据文件。此外，性能可视化与性能分析模块间的少量信息交换也遵循性能数据交换协议，以辅助性能文件的方式进行交换。

第三章 性能可视化模型

3.1 概述

随着第一台数字式计算机的诞生，就开始了计算机对数据进行可视化的研究。进行 80 年代以后，随着计算机处理信息量的不断增加，大量的数据使得可视化变得更加重要。1987 年在美国国家科学基金会(NSF)的图形图像专题讨论组的一份报告中首次引入了“科学计算可视化(Visualization in Scientific Computing)”一词。科学可视化的根本目标是把由实验或数值计算获得的大量数据转变成人的视觉可以感受到的计算机图像[9]。由于图像可以将大量的抽象数据有机地组织到一起，并能形象生动地展示数据所表现的内容以及其相互间的关系，可视化有助于人们对数据的理解和掌握。

性能可视化是科学可视化的一个分支，它是将并行应用程序的性能以可视的方式展现给用户的一种方法。本章将着重讨论性能可视化的抽象模型。这个模型以三层视图模型为中心，并以二级视图控制模型对性能可视化过程进行控制管理。

三层视图模型是性能可视化模型的主体。视图是表现并行应用程序性能的手段，每个视图表现了性能的一个侧面。三层视图模型是由性能数分析抽取层、性能数据可视化表示层以及可视化数据显示层组成。性能数据分析抽取层由性能数据集中抽取有效数据，解决可视化中“显示什么”的问题；性能数据可视化表示层将抽取所得的数据与某种可视形式联系起来，解决“如何显示”的问题；可视化数据显示层将已映射到某一视图格式上的可视化数据显示到物理设备上，从而实现视图的实际显示。这个三层模型将决定“显示什么”与“如何显示”的过程分开，将逻辑显示与物理显示的过程分开，使得与设备相关的部分仅限于模型的底层，从而提高了系统的可移植性和可扩展性。

视图是表现并行应用程序性能的主要方式。然而，视图只能表现出并行应用程序性能的一个侧面。为了能够更全面地反映并行应用程序的性能特性，性能可视化模型中建立了视图和视图管理器二级控制模型，即由视图级管理控制单一视图的显示过程，由视图管理器进行视图的全局控制，控制各个视图间的相关性、一致性，完成视图间相互参照、相互关联的功能。采用这种控制模型，一方面可以在单一视图中完成视图控制，另一方面又能提供视图的全局控制，使得系统易于扩充、改进。

3.2 三层性能视图模型

如图 4所示，三层视图模型由性能数据分析抽取层，性能数据可视化表示层和可视化数据显示层组成。

- **性能数据分析抽取层**

性能数据分析抽取层由性能数据集中获取性能数据，是进行可视化的第一步。这一层模型主要用于解决可视化的内容问题。一个视图通过这一层与性能监测模块交换信息，系

统则通过这一层来控制视图所表示的性能侧面。

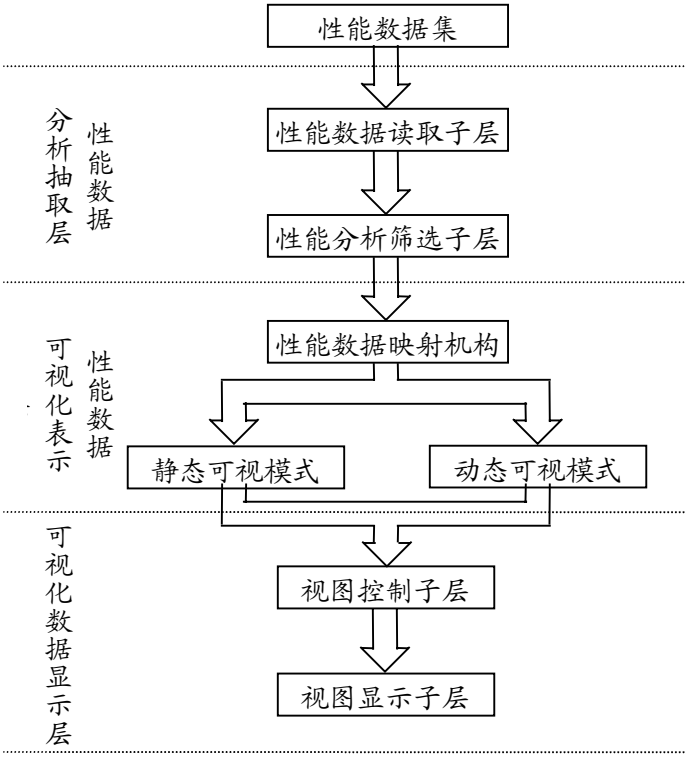


图4 三层视图模型

这一层的主要功能包括根据性能数据交换协议读取性能数据和分析筛选性能数据，形成性能数据子集。通过性能数据读取子层，性能可视化模块将性能数据由外部表示形式转换为内部表示形式。

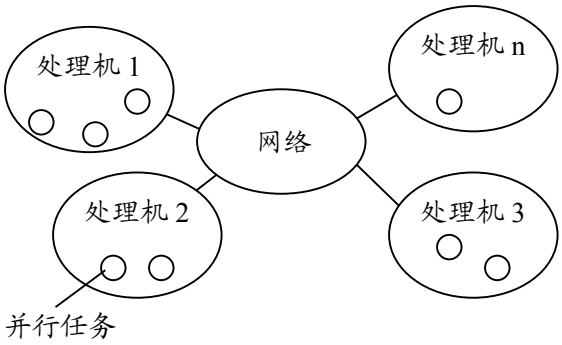


图5 并行任务分布示意

在基于消息传递机制的并行系统中，并行任务分布在不同的处理机上(如图 5所示)。任务间的消息传递可以不必考虑其所在处理机。一般地，相同处理机上的消息传递性能与

不同处理机间消息传递性能差别较大，因此，在进行性能分析时，需要掌握有关处理机和任务二级性能数据。为此，性能数据集的内部表示形式采用了二级模式，处理机集合(链表)记录处理机的有关信息，任务集合(链表)记录每一个任务的有关信息，每一个任务分别属于某一处理机，在处理机数据中有一指针指向其上任务集合(链表)。对于每一个任务记录其生命周期内的状态和它所发送或接收消息的情况。为了减少数据量，任务的状态以每一持续状态的范围的形式表示(即记录状态变化时刻)，消息则记录消息发送及接收时刻以及消息长度等有关信息。性能数据的二级表示形式如图 6所示。

性能分析筛选子层完成性能数据抽取分析的功能。性能数据抽取分析主要包括数据筛选、性能统计等不同方式。数据筛选是指从性能数据集中筛选出有效信息的过程，例如，筛选通信类性能数据、筛选某一处理机性能数据等。性能统计是在筛选的基础上对数据进行进一步的统计，它可以对全体数据进行统计，表现系统的统计性能；也可以对一些比较抽象的性能数据分时段进行统计，从而表现出性能变化的趋势。

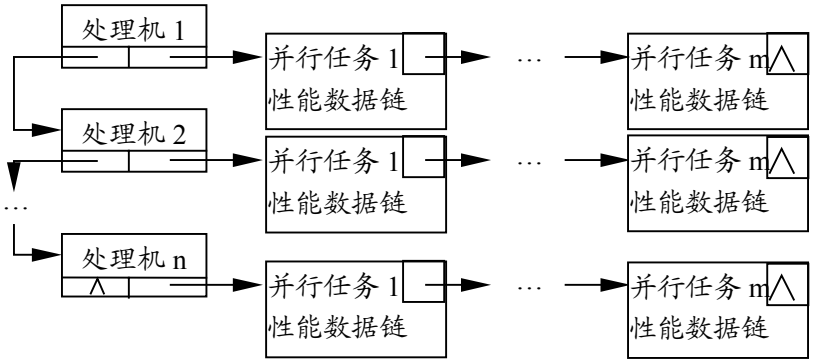


图6 性能数据的内部表示

- **性能数据可视化表示层**

性能数据可视化表示层由一个性能数据映射机构和若干视图模式组成。性能数据映射机构将筛选分析后的性能数据映射到不同的视图模式上，形成以可视形式表示的数据流(可视化数据)。视图可以分为静态类视图和动态类视图两大类模式。静态类视图将要表现的性能数据一次性显示出来，使程序员可以获得并行应用程序性能的总体特征。这类视图适于表现性能的统计类特征，主要形式有直方图、饼图、表格等。动态类视图反映了并行应用程序性能的变化过程，它按照时间顺序将程序运行中每一时刻(或时间区间)的性能显示出来，使程序员可以对系统性能的变化趋势有所了解。这类视图适于表现程序某一方面(如消息通信等)随着程序运行时间的推移而变化的情况。

- **可视化数据显示层**

可视化数据显示层由视图控制子层和视图显示子层构成。视图控制子层按照视图显示(播放)的要求，将上层形成的可视数据送给视图显示子层，由视图显示子层完成视图在物理设备上的显示。

视图控制子层由性能数据可视化表示层获得待显示的可视数据后，按照相应的视图模

式进行控制。对于静态类视图模式，其控制子层功能较为简单，主要包括视图的初始化及复位控制、数据查询控制及视图布局控制。对于动态类视图模式，其视图控制子层还包括视图播放控制及视图关联控制等。视图初始化及复位控制用于建立和关闭视图窗口；数据查询控制完成视图的在线数据查询；视图布局控制根据视图窗口大小自动布局显示比例，并在窗口大小变化时进行自动重新布局；视图播放控制是针对动态类视图的特殊控制，完成视图常速、快速、单步播放的控制功能；视图关联控制完成视图间相互参照、关联对比的控制。

视图显示子层按照视图控制子层的控制在物理设备上显示视图。这一子层将设备的物理特性对上层屏蔽，按照系统设置为上层绘制视图的几何原语赋以视觉特性，即确定视图的帧、颜色等，并启动视图绘制过程，完成视图显示。

3.3 二级性能视图管理机制

视图只表现了并行应用程序性能的一个侧面。为了能够全面地表现程序的性能，就需要使用多个视图共同表现程序的性能。各个视图相互关联、相互参照，形成程序完整的性能模型。为了能够更好地管理各个视图的显示与关联，我们采用了视图管理器和视图二级管理机制(如图 7所示)。

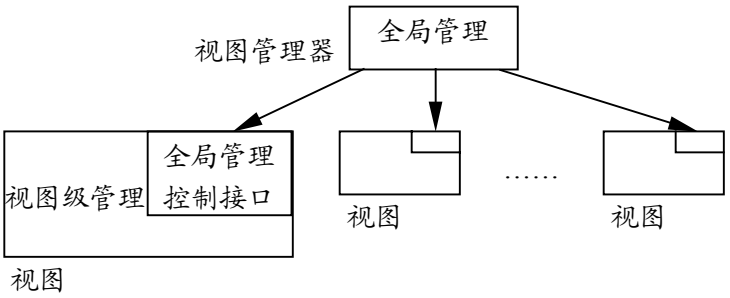


图7 二级性能视图管理机制示意

视图管理器用于视图的全局管理。它利用每个视图所提供的全局管理控制接口控制各个视图，以保证视图间的相关性、一致性。视图管理器从高层着手，在视图统一管理的同时，为用户提供了一个良好的用户界面，方便用户使用。视图级管理用于单一视图的显示控制。它是二级性能视图管理机制的内层。为外层视图管理器的全局管理提供一致的控制接口。

- **视图管理器**

视图管理器由视图注册/注销部件、视图显示控制部件和视图对象链表组成。视图注册/注销部件用于视图的登记注册和注销。一个视图打开后，首先要在视图管理器中注册；当关闭视图时，在视图管理器中注销相应的视图对象。一个视图从注册到注销称为该视图的生命周期。在视图的生命周期中，视图对象存放在视图管理器的视图对象链表中。视图管理器将根据视图对象链表中的视图对象对视图进行控制。

视图显示控制部件是视图管理器的主要组成部分。它通过已注册视图的控制接口对视

图进行全局控制。视图显示控制部件主要包括视图显示(播放)控制、视图关联控制和视图数据更新控制等。视图显示(播放)控制主要用于视图窗口的初始化/关闭、视图的显示控制等。视图关联控制是用于视图间信息关联的控制，它通过视图级控制所提供的视图关联定位接口进行控制。视图间的信息关联有根据关联点时间进行关联和根据关联点源代码进行关联二种。视图数据更新部件通过调用视图的数据更新接口强制各个视图更新其性能数据及刷新视图窗口。视图更新控制主要为性能监测模块提供更新接口，以便当性能监测模块获得新的性能数据集后及时更新各个视图。同时，用户也可以通过视图更新接口强制视图进行更新操作。

视图注册/注销部件	
视图控制部件	视图显示(播放)控制
	视图关联控制
	视图数据更新控制
	其它控制接口
视图对象链表	

图8 视图管理器

- **视图级管理控制接口**

与视图管理器相对应，视图提供了视图注册/注销与视图显示二类控制接口。视图注册/注销控制接口提供了视图注册(初始化)接口和视图注销接口。视图注册(初始化)接口用于初始化视图内部状态，注册视图及各控制接口函数；视图注销接口用于视图的注销及视图内部数据的清理释放功能。

视图注册/注销控制接口	
视图显示控制接口	视图窗口显示控制接口(display)
	视图帧刷新控制接口(refresh)
	视图复位控制接口(reset)
	视图缩放控制接口(zoom)
	视图关联定位控制接口(correlate)
	视图数据读取(筛选)控制接口(filter)

图9 视图级管理控制接口

视图显示控制接口提供了视图窗口显示、视图帧刷新、视图复位、视图缩放、视图关联定位及视图数据读取(筛选)等控制接口。除了视图数据读取(筛选)控制接口是在视图性能数据分析抽取层实现外，其它各个接口均在视图可视化数据显示层实现。

视图窗口显示(display)控制接口提供了视图窗口的初始化功能，它与视图帧刷新(refresh)控制接口、视图复位(reset)控制接口及视图缩放(zoom)控制接口共同完成视图的显

示功能。视图关联定位(correlate)控制接口用于视图间的关联定位；视图数据读取(筛选)控制接口(filter)用于视图数据更新控制。

第四章 VENUS 系统的设计与实现

4.1 概述

VENUS 系统是一个实用的性能可视化分析系统。它采用前面章节所设计的性能可视化分析框架模型，实现了一套用于并行系统上性能数据采集、可视化与分析优化的软件系统，为基于消息传递机制的并行程序设计提供了辅助开发、调试、优化工具。它的设计目标是实现一个独立于硬件平台及并行软件环境的并行应用程序的监测和可视化性能优化系统，目前所完成的 VENUS 系统已在工作站机群及曙光-1000 二种体系结构的并行平台上移植成功。

VENUS 系统是建立在性能可视化分析框架模型上的一个实用系统，实现了基于 PVM 消息传递机制的并行应用程序性能可视化分析。VENUS 系统的性能监测模块利用 PVM 内部跟踪机制，采用事件跟踪法对并行应用程序性能进行监测。为了增强对程序性能的监测能力，获取更多的性能信息，VENUS 系统采用了源代码修改与更改系统链接库相结合的方式，通过更改系统链接库，加强了系统的跟踪监测能力；通过源代码修改技术，为系统跟踪机制提供了更多的源代码信息并提供了对用户指定模块(代码段)特别跟踪监测的能力。

VENUS 系统性能可视化模块采用了二级视图管理和三层视图模型的机制，确定了视图的标准模型及接口，提高了系统的可扩展性。在视图的设计过程中，充分利用三层视图模型的特点，实现了性能视图在硬件拓扑结构上映射的能力，使性能视图更加直观。同时，VENUS 系统提供了强大的视图关联及源代码对应机制，在视图中能够很方便地获取相关信息。自定义事件时空图图示机制使用户能够很容易地对所关心的模块(代码段)进行特别监测，有利于程序细节部分的优化。此外，动态类视图显示细节度的自适应调整功能提高了动态类视图的播放速度，友好的用户界面和可用户化视图与界面控制方式为用户提供了一个良好的性能可视化分析环境。

VENUS 系统性能分析模块采用基于测量建立事件性能模型的方式，将并行应用程序性能数据与测量所得的性能模型进行比较，通过基于性能瓶颈假设验证的方式探测程序的性能瓶颈。假设验证采用可信度分析的方法进行模糊推理，避免了二值逻辑的缺点，提高了推理结果的可信度。

本章将着重论述 VENUS 系统的设计与实现。首先在第一节中分别介绍了 PVM、工作站机群及曙光-1000 的相关知识；然后在第二节中详细论述了 VENUS 系统的设计。第三、四两节分别论述了 VENUS 系统可视化部分以及辅助模块的实现。

4.1.1 PVM

4.1.1.1 PVM 简介

PVM(Parallel Virtual Machine)的研究开发最早始于 1989 年夏，它的开发队伍包括美国橡树岭国家实验室(ORNL)、Tennessee 大学、Emory 大学以及 CMU 等单位，并得到美国能源部、国家科学基金以及田纳西州的资助。PVM 是一套并行计算的系统软件，能够将不同体系结构的计算机，如工作站、并行机以及向量机等，通过网络连接起来，给用户提供一个一致的并行计算的软件平台[10]。

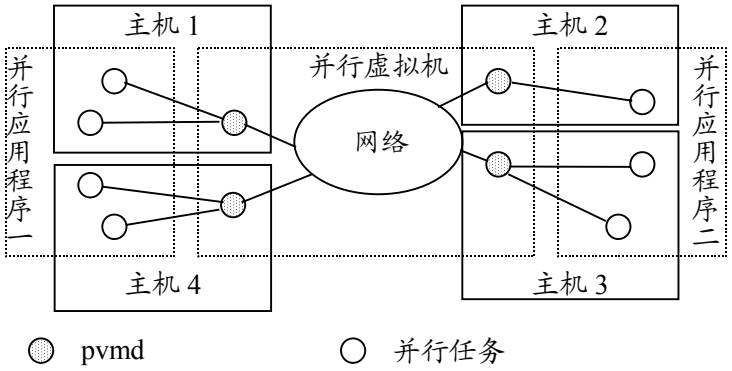


图10 PVM 并行虚拟机示意图

PVM 支持异种机互连，它通过软件的方法将不同种类体系结构计算机的硬件平台的区别屏蔽起来，为应用级程序员提供了一个统一的基于消息传递(Message Passing)的并行环境。PVM 将连接在一起的计算机(处理机)称为主机(Host)，在主机上运行的 PVM 并行进程称为任务(Task)。系统的消息传递机制是通过运行于系统各个主机上的 pvmd(PVM Daemon)完成的。运行于各个主机上的 pvmd 构成了一个虚拟并行系统，程序员可以完全不关心各主机的不同而将并行程序(任务)分配到这个虚拟并行系统之中。

在图 10中，中间虚线框内由网络连接起来的四个 pvmd 组成并行虚拟机，对用户(程序员)透明。并行应用程序运行于这个并行虚拟机之上，每个并行应用程序可以分布在整个虚拟机上；在并行虚拟机上可以同时运行多个并行应用程序。每个主机上有一个 pvmd，它用于管理本机上的并行任务，并负责并行任务间的通信。

一般情况下，PVM 通过 pvmd 进行任务间的通信。对于高版本的 PVM 还有一种可伸缩性稍差但速度更快的传送法，它允许 PVM 在两个任务之间建立一条直接的任务到任务的 TCP 链路。最初的 TCP 建立时间很长，但是所有后来的消息在相同的两任务间比缺省的发送方法快 2-3 倍。这种方法的主要缺陷是每个 TCP 通信端点使用一个文件描述符。因此，当虚拟机上任务的数量为 n 时，就可能需要 $O(n^2)$个文件描述符[11]。

PVM 的消息传递机制分为两层。第一层是系统消息收发层，使用 UDP 传输数据进行 pvmd 间的通信，主要任务是进行各主机之间的协调。其上流动着两类消息：一是 pvmd 间的控制消息，用于完成各类 PVM 内部管理及控制功能；另一类是由 pvmd 转发的任务间的消息。在此基础上建立了用户任务消息收发系统，它建立在 TCP 通信管道的基础上，

在系统消息收发层的帮助下实现 PVM 任务间的通信。

一个典型的 PVM 服务如图 11所示：用户任务调用 PVM 库函数要求提供某种服务，PVM 库函数向系统的本地 pvmd 发出一个与请求相应的 TM 类消息，请求提供这项服务。本地 pvmd 将此请求用对应的 DM 类消息转给有能力处理此请求的 pvmd(可能是它自己)，收到 DM 消息的 pvmd 就执行这个工作，把查询或执行的结果回送给本地 pvmd，这个 pvmd 再把这个结果回送给提出服务的任务。

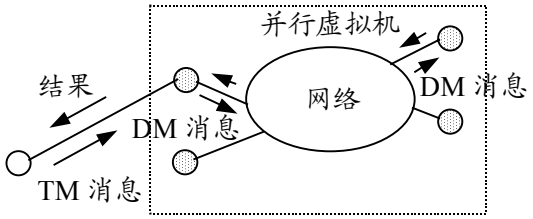


图11 一个典型的 PVM 服务

PVM 的消息发送有两种方式，一是阻塞式消息发送，一是非阻塞式消息发送。无论是哪一种发送方式，在消息发送的过程中，消息都将暂存在消息缓冲区中。消息通过打包指令，转换成网上格式并存入消息缓冲区，等待发送；接收到的消息也要暂存在这里，经解包后再传送给用户任务。消息缓冲区从功能上分为发送缓冲区、接收缓冲区，分别用于发送消息和接收消息的缓存；从它存在的层次上又分为系统消息缓冲区和用户消息缓冲区，系统消息缓冲区用于缓存系统级的消息(用于系统通信链路的建立、维护等用途的消息，对用户透明)，用户消息缓冲区则用于缓存用户程序中所产生的消息。用户消息缓冲区可以有多个，而系统消息缓冲区只能有一个[12]。

4.1.1.2 PVM 的跟踪机制

PVM 系统内部提供了一个基于事件的跟踪机制，它可以根据用户的要求，对系统中与 PVM 有关的重要事件发生的时间及其相关信息记录下来。与 PVM 有关的事件主要有以下二类：(1)PVM 函数调用事件，这类事件记录了调用了 PVM 系统函数过程的相关信息(包括该函数的起止时刻以及相关的一些信息)，一般情况下 PVM 是通过二个事件记录来描述这一类事件；(2)PVM 系统事件，包括 newtask、endtask、spawntask 三个，它们用于记录 pvmd 在执行 PVM 功能调用时并行虚拟系统状态变化的相关信息(包括系统状态、发生状态变化的时刻及其它信息)，这类事件是通过记录系统状态发生变化的时刻来描述系统的状态，因而只需产生一个事件记录。

PVM 内部在库函数的实现过程中加入了跟踪机制，在 PVM 环境下(PVM 应用程序中或 PVM 的控制台上)只需使用 pvm_setopt()函数设置跟踪数据的采集进程和一些其它的相关参数，就可以在数据采集进程中获得所需的性能数据。

下面的程序列出了一个标准的 PVM 函数的结构。程序通过全局变量 pvmtoplvl 判断是否需要跟踪，并通过宏 TEV_DO_TRACE 进行一步判断本函数是否需要跟踪。最后通过宏 TEV_FIN 完成事件的记录过程。在函数体的最后，仍是通过上述二步判断来决定是否记录终止事件。

```
        ……
/* 判断是否需要跟踪，如果需要跟踪，则记录起始事件 */
if (x = pvmtoplvl) {
    pvmtoplvl = 0;
    if (TEV_DO_TRACE(TEV_XXX0))      /* 判断是否需要跟踪 */
        TEV_FIN;                     /* 记录一个事件 */
}
        ……
/* 函数实现过程 */
        ……
/* 判断是否需要跟踪，如果需要跟踪，则记录终止事件 */
if (x) {
    if (TEV_DO_TRACE(TEV_XXX1)) {
        pvm_pkint(&cc, 1, 1);        /* 判断是否需要跟踪 */
        TEV_FIN;                     /* 记录一个事件 */
    }
    pvmtoplvl = x;
}
        ……
```

4.1.2 系统的硬件平台

4.1.2.1 工作站机群(NOW)的结构

清华可扩展工作站机群是由八台 SUN SPARC 20、八台曙光天演工作站及若干台 SUN SPARC Ultra 等其它工作站或服务器通过高速以太网连接构成的异构 Cluster 结构，其并行环境由标准的 PVM 系统提供。

4.1.2.2 曙光 1000(MPP)的结构

中科院计算所的曙光 1000 并行机的运行环境包括五个部分：结点阵列、控制台、主机(Horn)、前端机(Peach)和终端工作站。

结点阵列是由 32 个计算结点和 4 个 I/O 结点构成的一个虚拟机器(MPP)。每一个计算结点相当于一个独立的处理机，带有一块 i860 微处理器和 32MB 的内存。计算结点可以组织在一起，运行同一应用程序，并通过消息传递来交换数据。I/O 结点具有 I/O 接口，负责管理曙光 1000 系统的磁盘和其它 I/O 设备，通过互连网络与其它结点通信。用户只能通过主机(Horn)对它们进行操作。

控制台是一台 PC 486，运行 SCO UNIX 操作系统，负责监控和诊断计算结点，也可以用来测试计算结点的硬件。它对用户是透明的。

主机(Horn)是一台基于 EISA 总线的 80486 或其它 PC 机，运行 MACH 2.6 或 LINUX 操作系统，负责管理整个系统，并向用户提供并行任务加载等服务。

前端机(Peach)是一台 SUN SPARC 2 工作站，它是用户操作的主要平台，用户的程序一般存放在它上面，并在它上进行编译(交叉编译)等。

终端是用户上机所使用的平台(机器)，在曙光 1000 上所使用的终端是 SUN SPARC 2，它的主要功能是为用户提供一个图形终端，一般情况下，用户程序并不放在终端机上。可以把终端看成有少量硬盘空间的 X 终端。

4.2 系统总体设计

4.2.1 性能数据交换协议

4.2.1.1 PVM 中并行程序的状态模型

一个并行程序通常是通过将一个问题(任务)分解映射到多个处理机(进程)之上。一个问题分解为多个可并行解决的问题通常有两种方法，一是采用定义域分解技术将问题按照定义域均匀地划分为多个任务。这类问题一般采用静态分解的方式，首先将数据定义域分布给各个结点，然后限制计算，使每个结点程序只更新其上的数据子定义域，最后通过结点间通信，将结果汇总。

问题分解的另一种方法是控制分解技术。当问题的定义域和数据结构都不太规则或不可预测时，就不能应用定义域分解技术。控制分解技术把一个算法看作是一组相互连接的功能模块。对于小型问题来说，这些功能模块将被一个接一个地顺序执行，但是大型问题可能在模块间有明显的重叠。控制分解技术将不同的功能模块分配给不同的处理机结点，不同的功能将在不同的结点上执行[13]。

在 PVM 中，定义域分解技术能常采用 SPMD 模式实现，控制分解技术通常采用 Master/Slave 模式实现。但无论是 SPMD 这种对称的代码结构，还是 Master/Slave 这种非对称的代码结构，都是通过并行应用程序的一个并行子任务(主任务)启动其它相同(SPMD)或不同(Master/Slave)的程序(任务)实现的。这个启动其它任务的任务是第一个注册进 PVM 环境的任务*，在 VENUS 系统称之为主任务(Master Task)；其它由主任务派生的任务，称为一般任务(General Task)。这样区分仅是从称谓方便考虑，并未限制任务间的主从关系。实际上，一个 PVM 应用程序的所有并行子任务可以完全对等地、异步地执行。

很显然，整个并行程序的执行周期应从主任务的起始到所有并行子任务的全部结束为止。对于一般任务而言，它们是由主任务或其它一般任务通过 pvm_spawn()派生的。PVM 任务被派生时由系统自动将其注册入 PVM 环境。而 PVM 本身的 Trace 机制在新的任务注册之时会产生一条 newtask 事件，因此一般任务可以以 newtask 事件作为任务起始标志。对于主任务则不能直接将 newtask 事件作为任务起始标志。因为在主任务中第一次使用 PVM 函数注册入 PVM 环境之前，可能还有一些准备工作需要串行执行。对此，VENUS 系统的代码插入模块应在主任务的 main()函数的第一条执行语句之前插入 pvm_mytid()或其它无副作用的函数调用，以此强制在主任务的起始产生一条 newtask 事件作为任务开始标志。这样做是必需也是必要的：VENUS 的代码插入机制产生的事件总是要通过 PVM 传递给性能数据接受者的。

并行任务的终止时间比较容易确定。PVM 的 Trace 机制保证在每一任务结束时产生一

* 在 PVM 应用程序中，由主任务第一次调用 PVM 函数时将其注册为一个 PVM 任务，其它一般任务则是在生成该任务时就注册为 PVM 任务。

条 endtask 事件，此事件即可作为主任务和一般任务的结束标志。需要指出的是，这里的任务终止是指任务进程的终止，而不是调用 pvm_exit()的结果，后者仅是将进程从 PVM 环境中注销(即该进程不再作为 PVM 的一个任务，进程仍然存活)。

并行任务在其生存周期中可处于以下几种状态：

- **忙状态(Busy)**

忙状态是指并行任务执行有效计算的状态。有效计算很难给予精确的定义，它往往是和具体应用相关的。一般认为有效计算是指用于计算原来串行任务的时间，即扣除因并行化而增加的代码后的代码执行时间。在 PVM 中，并行任务加载后的初始状态为忙状态。

- **空闲状态(Idle)**

空闲状态是指并行任务因同步等待接受消息或其它原因而使处理机处于空转的状态。这一状态包括由于阻塞而造成的处理机空转，以及由于一个任务执行完毕而其它任务尚未执行完毕而造成的处理机空转。

- **开销状态(Overhead)**

开销状态是指处理机用于执行由于并行化而增加的代码的状态。将一个应用程序并行化后，将增加一些由于并行计算而必须的同步指令，此外，还有为完成同步指令而增加的其它代码。在 PVM 中，用于同步的 pvm_barrier()、pvm_recv()等函数会产生同步等待，而其它的 PVM 函数也会产生一些用于计算以外的开销，这些都属于额外开销。

4.2.1.2 PVM 中并行程序状态转换的有限状态机模型

在 PVM 中，并行程序在任一时刻都处于忙(Busy)、空闲(Idle)或开销(Overhead)三种状态之一。VENUS 系统中采用了事件跟踪法来获取并行应用程序的性能数据。为了能够通过采集到的性能数据(事件记录)推导出并行程序所处的状态，在 VENUS 系统中建立了 PVM 并行程序状态转换的模型，如图 12所示。

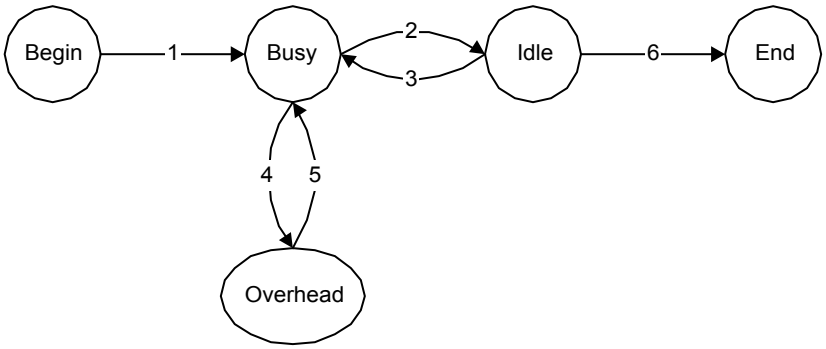


图12 PVM 并行程序状态转换的有限状态机模型

- **并行任务的起始终止和忙状态**

在 VENUS 系统中，性能监测模块所采用的机制能够保证，无论是主任务还是一般任务，在其起始都将向或由 PVM 系统自动完成任务在 PVM 环境中的注册，同时，PVM 在每一个并行任务结束时会产生一个终止事件(endtask 事件)。因此，在通过分析每一个并行任务的起始事件(newtask 事件)和终止事件就可以获得并行任务的起始与终止时刻。

并行任务在起动后处于忙状态。忙状态是一个相对的稳定状态，在系统发生某些事件后，任务切换至另两个状态，一段时间(系统发生对应的事件)后会再次恢复为忙状态。

- **空闲状态的转换**

空闲状态是由于通信、同步等原因造成的处理机空转的状态。除了上述由 endtask 事件引起的向空闲状态转换外，并行任务转换至空闲状态一般是由于任务间的同步等待引起的。在 PVM 应用程序中，同步等待通常包括三类函数调用：

- ✧ 同步消息接收函数，包括 pvm_recv()，pvm_trecv()等；
- ✧ 组同步函数，包括 pvm_barrier()，pvm_freezegroup()，pvm_gather()，pvm_scatter()，pvm_reduce()等；
- ✧ 系统调用，如 sleep()等。

这些函数将产生入口和出口二个事件，入口事件使得任务由忙状态转换为空闲状态，出口事件使得任务由空闲状态转回忙状态。

- **通信事件及开销状态**

理论上，对 PVM 函数的调用在原来串行程序中是没有的，因此可以认为 PVM 函数调用是程序并行化增加的开销。但这样做势必将导致必须监测所有的 PVM 事件以获得必要的任务状态信息，其弊端有三：在监测大应用程序时可能导致超大容量的性能数据集；频繁的事件记录将使程序的失真校正困难化；性能监测机制本身成为系统性能瓶颈，不利于以后的分析。

基于上述考虑，在 VENUS 系统中只选择了部分主要的事件作为导致转换为开销状态的原因。这些事件是由在 PVM 程序中发生的频率效高，且开销较大的函数引起的。引起状态转换为开销状态的函数有：

- ✧ 消息发送类函数，包括 pvm_send()、pvm_mcast()和 pvm_bcast()；
- ✧ 消息异步接收类函数，包括 pvm_nrecv()；
- ✧ 消息缓冲区管理类函数，包括 pvm_initsend()、pvm_mkbuf()和 pvm_freebuf()；
- ✧ 消息缓冲区打包类函数，包括 pvm_packf()、pvm_pkbyte()、pvm_pkcplx()、pvm_pkdcplx()、pvm_pkdouble()、pvm_pkfloat()、pvm_pkint()、pvm_pkuint()、pvm_pkshort()、pvm_pkushort()、pvm_pklong()、pvm_pkulong()、pvm_pkstr()；
- ✧ 消息缓冲区解包类函数，包括 pvm_unpackf()、pvm_upkbyte()、pvm_upkcplx()、pvm_upkdcplx()、pvm_upkdouble()、pvm_upkfloat()、pvm_upkint()、pvm_upkuint()、pvm_upkshort()、pvm_upkushort()、pvm_upklong()、pvm_upkulong()、pvm_upkstr()；
- ✧ 任务调度类函数，包括 pvm_spawn()和 pvm_exit()。pvm_spawn()和 pvm_exit()函数涉及 PVM 内部的并行任务调度。

4.2.1.3 性能数据集和性能数据交换协议

性能数据交换协议定义了性能数据集中事件记录的规范和类型。VENUS 系统采用与 XPVM 兼容的 SDDF(Self-Describing Data Format)格式的跟踪文件(Trace File)作为性能数据集的实现形式。SDDF 是一种自解释的变长记录格式的 ASCII 文件[14]。一般地，PVM 为其每一库函数预定义了两种事件：入口事件和出口事件。例如在跟踪机制下，应用程序对 pvm_exit()函数的调用将产生两条事件记录：exit0 和 exit1。一条事件记录在 SDDF 中占一

行，不同的事件记录的内容可能不同，但每一事件记录一般都包含事件类型名及编号、时间戳、产生该事件的任务号以及事件发生的源代码位置(模块名及行号)等，如下所示：

```
#15:"exit0"                          /* 事件类型名及其编号 */
{
    // "Time" "Second"
    int "sec";                       /* 时间戳，秒部分 */
    // "Time" "Micro Second"
    int "usec";                      /* 时间戳，微秒部分 */
    // "TID" "Task ID"
    int "tid";                       /* 产生该事件的任务号*/
    ……
    // "Mn" "Module name"
    char "Mn"[];                     /* 产生该事件的模块名 */
    // "Ln" "Line Number"
    int "Ln";;                       /* 产生该事件的位置在模块中行号 */
};;
```

4.2.1.4 SDDF 解释器

SDDF 格式的事件记录是自解释的，亦即事件的格式是灵活可变的。为保持这种灵活性，SDDF 解释器仅提供低一层的事件解释。VENUS 系统中事件记录内部表示如下所示：

```
typedef struct _Args {
    char name[MAX_NAME_LEN];      /* 参数名 */
    char type;                    /* 参数值类型 */
    char value[MAX_VALUE_LEN];    /* 参数值 */
} Args;                           /* 参数结构 */
typedef struct _EventSDDF {
    int no;                       /* 事件的编号 */
    char event[MAX_NAME_LEN];     /* 事件名称 */
    int argc;                     /* 事件记录的参数数 */
    Args args[MAX_ARGS];          /* 参数数组 */
    struct _EventSDDF * next;     /* 链表指针 */
} EventSDDF;
```

参数的值为字符串形式，调用者需根据其类型采用相应的类型转换函数进行转换。

SDDF 解释器提供的主要接口如下：

- **GetEventSDDF(FILE *fp, EventSDDF **ev)**

该过程扫描性能数据文件 fp。在扫描跟踪文件的过程中，每碰到一条 SDDF 事件的描述，就创建一个 SDDF 事件结构，根据 SDDF 事件描述填写事件结构的各个参数域(此时事件结构的每个 Args 参数结构的 value 值是空白的)，并通过链表指针加入到 SDDF 描述库中。当扫描至一条事件记录时，在 SDDF 描述库中查找与其匹配的描述，若找到则填写

该事件结构各 Args 参数的 value 值，并且在参数 ev 中返回指向事件结构的指针。当参数 ev 的返回值为 NULL 时，表示搜索到文件尾也未找到正确的时间记录。该函数的返回值表示操作成功与否：SDDF_OK 表示找到了一条事件记录，SDDF_EOF 表示文件已搜索完毕。

- **EventType(EventSDDF *ev)**

  返回事件的类型。VENUS 系统中将 PVM 事件分为以下几类：

  HOST_EVENT：主机类事件，包括 host_add、host_del 和 host_sync 事件

  TASK_EVENT：任务类事件，包括 newtask 和 endtask 事件

  OUTPUT_EVENT：I/O 事件，包括 output 事件

  ASEND_EVENT：单一消息发送事件，包括 send0 和 send1 事件

  BSEND_EVENT：广播消息发送事件，包括 bcast0、bcast1、mcast0 和 mcast1 事件

  RECV_EVENT：阻塞式消息接受事件，包括 recv0、recv1、trecv0 和 trecv1 事件

  NRECV_EVENT：非阻塞式消息接受事件，包括 nrecv0 和 nrecv1 事件

  GROUP_EVENT：组事件，包括，包括 joingroup0、joingroup1、lvgroup0 和 lvgroup1 事件

  BARRIER_EVENT：组同步事件，包括 barrier0 和 barrier1 事件

  COMMON_EVENT：其它事件

- **EventStatus(EventSDDF *ev)**

  返回事件的对应的程序状态。可以为忙(STATE_BUSY)、开销(STATE_OVERHEAD)和空闲(STATE_IDLE)三者之一。

- **char * EventSDDFGetValue(EventSDDF *ev, char *name)**

  返回事件的名为 name 的参数的值 value。

4.2.2 工程化管理

4.2.2.1 工程化管理模式

VENUS 系统中采用了工程化管理模式，对一个并行程序性能进行分析优化的过程称为一项工程(project)，它描述了并行程序的代码信息(包括源代码、Make 文件、可执行代码等)、执行情况、性能分析结果及优化建议等。通过工程的方式对并行应用进行性能分析优化，使得分析优化过程更加清晰明了，便于管理。

工程主要包括以下几个部分：

- **代码信息**

  这一部分描述了并行程序的静态信息。它包括由用户(程序员)指定的源代码、Make 文件、可执行代码以及由 VENUS 系统分析所得有关代码结构的信息。

- **性能数据**

  这一部分记录了系统的执行情况。在 VENUS 系统中，并行应用程序的执行情况由相应的跟踪文件表示。

- **分析结果**

  这一部分记录了对性能数据进行分析后所得的结果以及所提出的优化建议。

- **配置信息**

  这一部分保存了该工程的有关配置，其中包括视图配置、跟踪与优化配置等。

4.2.2.2 工程文件格式

工程文件的格式如图 13所示，工程文件头标志和版本号用来检查工程文件的格式，目前支持的版本为 1.00。为了防止用户手工修改对工程文件造成的破坏，加密字节后的部分全部用加密字节进行异或运算变成不可打印的 ASCII 字符。资源文件名段给出了工程的资源文件的路径名，跟踪文件名段给出了工程的性能数据文件的路径名。每一个工程项目包含两部分：工程项目类型和项目文件名。工程项目类型可以为 Source、Makefile 和 Executable 之一。

!VENUS			工程文件头标志(6 bytes)
Version			版本号(2 bytes)
Magic Number			加密字节(1 bytes)
工程信息	Title		工程的标题(字符串)
	Description		工程简单描述(字符串)
	Manager		工程管理者(字符串)
	Organizer		工程所有者(字符串)
	Type		工程类型(字符串)
	Keyword		工程检索关键字(字符串)
	Comments		备注信息(字符串)
Resource File			资源文件名(字符串)
Trace File			跟踪文件名(字符串)
工程项目信息	type	name	工程项目 1 类型(字节)、名称(字符串)
	type	name	工程项目 2 类型(字节)、名称(字符串)
	…	……	
	type	name	工程项目 N 类型(字节)、名称(字符串)

图13 工程文件格式

4.2.2.3 工程数据结构

与工程文件相对应，当用户打开一个工程时，需要为其创建内部的数据表示。工程数据结构定义如下：

```
typedef struct _ProjectItem {
    char type;                    /* 项目类型 */
    char name[BUFSIZE];           /* 项目文件名 */
    struct _ProjectItem * next;   /* 链表指针 */
} ProjectItem;                    /* 工程项目结构 */
```

```
typedef struct _Project {
    char title[BUFSIZE];          /* 工程标题 */
    char descript[BUFSIZE];       /* 工程描述 */
    char manager[BUFSIZE];        /* 工程管理者 */
    char orgnize[BUFSIZE];        /* 工程所有者 */
    char type[BUFSIZE];           /* 工程类型 */
    char keyword[BUFSIZE];        /* 工程检索关键字 */
    char comments[BUFSIZ];        /* 工程备注 */
    char prj_file[BUFSIZE];       /* 工程文件名 */
    char res_file[BUFSIZE];       /* 工程资源文件名 */
    char dat_file[BUFSIZE];       /* 工程跟踪文件名 */
    int nitems;                   /* 工程项目数 */
    ProjectItem *items;           /* 工程项目链表 */
    int nhosts;                   /* 程序运行中所利用的主机数 */
    HostList hosts;               /* 主机列表 */
    int ntasks;                   /* 程序运行中产生的并行子任务数 */
    TaskList tasks;               /* 任务队列 */
    int ngroups;                  /* 程序运行中所产生的任务组数 */
    GroupList groups;             /* 任务组队列 */
    struct timeval app_start;     /* 程序起始时间 */
    struct timeval app_end;       /* 程序终止时间 */
    struct _Project *next;        /* 链表指针 */
} Project;
```

工程结构中维护了主机列表、任务队列、任务组队列等程序并行资源。当打开工程时，需要根据工程文件和 Trace File 的内容填充工程结构的各个域，而在关闭工程时则要释放工程结构中各个域占用的内存。系统中定义一个全局的工程结构指针 project。

4.2.3 主要模块的设计

4.2.3.1 性能监测模块的设计

性能监测模块利用 PVM 内部的性能跟踪机制，采用源代码修改和更改系统链接库相结合的方式实现。通过修改源代码使性能监测部件可以获取更多的并行应用程序性能数据；通过更改系统链接库，扩展了 PVM 原有的跟踪机制，增强了对程序性能信息的获取能力。这部分的具体设计实现是由课题组其它成员完成，详细资料可参阅相关文章[15][16]。

4.2.3.2 性能可视化模块的设计

性能可视化模块采用了三层视图模型、二级视图控制模型。在视图管理器的统一管理下，每个视图通过内部的性能数据分析抽取层、性能数据可视化表示层将性能数据映射为可视化数据，并通过可视化数据显示层将可视化数据映射到物理设备上。为了便于移植，与设备无关的性能数据分析抽取层和性能数据可视化表示层采用 C 语言设计，而可视化数据显示层采用 Tcl/Tk 语言设计[17]。

视图是用于表现并行应用程序性能的工具。因此，视图内容与形式的设计是性能可视化模块中的重要部分。VENUS 系统中的视图从内容上分为四类：系统信息类视图、利用率类视图、通信类视图和并行任务类视图。系统信息类视图表现系统静态信息，主要包括系统硬件及任务分配情况等；利用率类视图表现系统处理机、任务等的利用率信息；通信类视图表现任务、处理机间的通信情况；并行任务类视图表现并行任务中用户指定对象的性能情况。

VENUS 系统中的视图从表现形式上分为三类：静态类视图、动画类视图和历史记录类视图。静态类视图主要以图表形式表现；动画类视图以连续的帧的形式表现系统的性能状态；历史记录类视图以时间为坐标记录系统性能的变化情况。

可视化是以图形的方式表示数据。因此，图形显示的速度是各类可视化系统必须解决的问题。在 VENUS 系统中采用了视图播放细节度动态自动调节的技术，使视图显示的速度大大提高。此外，视图内容随视图窗口大小变化而动态调整也是视图实现过程中较难控制的问题。视图播放细节度动态自动调节技术及视图内容的自动布局控制将在下面的章节中进行讨论。

视图管理器是视图显示(播放)的控制机构。在系统内部，视图管理器需要记录并管理各个视图；在用户界面中，视图管理器体现为视图播放控制面板(Play Panel)。用户通过这个控制面板来控制各个视图的显示(播放)及相互关联。

有关视图及视图管理器的实现将在下一节中详细讨论[18]。

4.2.3.3 性能分析模块的设计

性能分析模块分为三个部分：性能建模部分、假设验证部分和综合分析部分。性能建模是通过一组基准测试程序对系统硬件进行测试，并建立一定的性能模型。假设验证通过性能建模部分所建立的性能模型与性能数据的对比计算各个瓶颈的可信度，从而验证并行应用程序的瓶颈假设。综合分析通过对瓶颈可信度以及其它性能信息的分析给出并行应用程序的性能评价与改进建议。

在性能分析的过程中，性能模型的选择是一个重要的环节。在 VENUS 系统中采用了根据性能尺度(Metrics)的上下界与阈值拟合性能模型的方式，根据大数定理，将性能可信度函数拟合为指数函数的形式。通过实测，这种方式对性能瓶颈假设的验证准确性较高。

VENUS 系统性能分析模块的具体实现是由课题组其它成员完成，详细资料可参阅相关论文[19]。

4.3 性能可视化模块的实现

4.3.1 性能数据的内部表示

4.3.1.1 事件

程序行为再现的主要依据是记录在跟踪文件中的性能事件，每一性能事件都表示某一程序行为的发生，从而导致程序状态的变迁。因此首先需要为事件建立内部表示。由于每一并行任务的执行都可以看成一系列状态的变化，因而 VENUS 系统中采用状态链表来表示任务执行的全过程，而每一个状态结点也包含了导致进入该状态的事件。

VENUS 系统中定义如下的状态结构：

```
typedef unsigned long EMask;
typedef unsigned long Description;

typedef struct  {
    unsigned long module;        /* 模块号 */
    unsigned long line;          /* 行号 */
} SourceLocation;

typedef struct _State {
    int status;                  /* 状态，可为忙、开销或空闲之一 */
    struct timeval start_time;   /* 状态起始时间 */
    struct timeval end_time;     /* 状态结束时间 */
    Emask event;                 /* 引发状态的事件号 */
    Item id;
    SourceLocation src_loc;      /* 事件在程序中的定位信息 */
    Description descript;
    struct _State *next;         /* 状态链指针 */
} State, *StateQueue;
```

src_loc 成员与 SDDF 事件的最后两个域对应。在扫描跟踪文件时，为每一事件创建对应的 State 结构，而每一任务的所有状态结构将构成一个单向链表。

4.3.1.2 消息

对于 PVM 应用平台而言，原子事件一般表示一个 PVM 函数的调用。为了减少内存开销，函数调用的参数信息没有在状态结构中保留(系统提供特定的视图查看每一事件的详细信息)。对于 PVM 通信函数调用的事件，除了创建相应的状态结构外，还需保留有关的消息参数。VENUS 系统中定义消息结构如下：

```
typedef struct _Msg {
    int tag;                     /* 消息结构标志 */
    int src_tid;                 /* 发送消息的源任务号 */
    int dst_tid;                 /* 接受消息的目的任务号 */
    int msg_tag;                 /* PVM 消息标志 */
    int nbytes;                  /* 消息长度，字节数 */
    struct timeval sendtime;     /* 消息发送时间 */
    struct timeval recvtime;     /* 消息接受时间 */
    Item id;
    State * s_state;             /* 源任务产生消息的事件状态 */
    State * d_state;             /* 目的任务接收消息的事件状态 */
    struct _Msg * send_next;     /* 发送消息队列的下一个 */
    struct _Msg * recv_next;     /* 接受消息队列的下一个 */
```

```
} Message, *MessageQueue;
```

每一消息结构记录了一次通信。可以保证每一消息通信的源和目的任务均是唯一的(广播消息发送可以分解为多个单一消息发送看待),因此可以唯一确定与每一消息结构相联结的两个任务的状态结点。在扫描至消息发送事件时，首先在通信的源任务的状态链表中增加一个状态结点，然后创建相应的消息结构。当扫描至目的任务的某一消息接受事件，且该接受事件与此消息结构相匹配时，即可返填该消息结构的 d_state 域，使其指向接受事件对应的状态结构。

消息结构的 tag 域表示消息通信事件的类型，可为以下三种值：

```
ASEND_EVENT          单一消息发送事件
BSEND_EVENT          广播消息发送事件
BARRIER_EVENT        组同步事件
```

每一消息结构联结着两个状态点：发送任务的状态点和接受任务的状态点。VENUS 系统为每一任务维护一个发送消息队列(单向链表)，该任务的消息发送事件所产生的消息结构通过 send_next 域加入该队列中。同时 VENUS 系统还为每一任务维护一个接受消息队列，每一消息结构创建时不仅要加入发送任务的发送消息队列，还要加入接受方的接受消息队列。

4.3.1.3 任务

VENUS 系统中定义如下的数据结构来描述每一并行任务：

```
typedef struct _Task {
    char name[MAX_NAME_LEN];          /* 任务对应的可执行程序名 */
    int tid;                          /* 任务号 */
    int ptid;                         /* 任务的父任务号 */
    int hostid;                       /* 任务所在主机号 */
    int index;                        /* 任务的顺序号 */
    int y1,y2;
    struct timeval start_time;        /* 任务的起始时间 */
    struct timeval end_time;          /* 任务的结束时间 */
    struct timeval d_time;            /* 任务时钟同步差 */
    struct timeval user_time;         /* 任务消耗的用户时间 */
    struct timeval sys_time;          /* 任务消耗的系统时间 */
    StateQueue states,s_tail;         /* 任务的状态队列 */
    MessageQueue s_msgs,s_msgs_tail; /* 任务的发送消息队列 */
    MessageQueue r_msgs,r_msgs_tail; /* 任务的接受消息队列 */
    struct _Task *next;               /* 任务链指针 */
} Task, *TaskList;
```

d_time 域表示该任务与主任务进行同步时钟校正的差值，可为正为负。除了记录任务的有关信息外，任务结构的最重要部分是它维护三个队列，且所有并行任务结构也通过链指针构成一个任务队列。

4.3.1.4 任务组

定义如下结构来描述 PVM 任务组：

```
typedef struct _Group {
    char name[MAX_NAME_LEN];      /* 任务组名 */
    int ntasks;                   /* 任务组包含的任务数 */
    int *tids;                    /* 任务组包含的各任务号 */
    struct timeval *start;        /* 各任务加入任务组的时间 */
    struct timeval *end;          /* 各任务离开任务组的时间 */
    struct _Group *next;          /* 链表指针 */
} Group, *GroupList;
```

4.3.1.5 主机

定义如下结构来描述每一主机情况：

```
typedef struct _Host{
    char name[MAX_NAME_LEN];      /* 主机名 */
    char IPAddress[MAX_NAME_LEN]; /* 主机 IP 地址 */
    char OS[MAX_NAME_LEN];        /* 主机操作系统 */
    int hostid;                   /* 主机号 */
    int speed;                    /* 主机相对速度 */
    int ntasks;                   /* 程序在该主机上派生的任务数 */
    int *tids;                    /* 在该主机派生的各任务号 */
    struct timeval d_time;        /* 该主机的同步时钟校正值 */
    struct _Host *next;           /* 链表指针 */
} Host, *HostList;
```

4.3.2 视图管理器

4.3.2.1 视图管理器的内部表示

视图的一系列函数指针成员表示可对视图施加的操作，它们是为了对视图进行统一管理而设立的。为此设置视图管理器对象结构如下：

```
typedef struct {
    ParallelArch parch;           /* 并行系统结构信息 */
    PDataQueue pdata_spool;       /* 性能数据缓冲区 */
    PlayPanel ppanel;             /* 视图播放控制面板 */
    int nviews;                   /* 系统中注册的视图数 */
    ViewList views;               /* 视图链表 */
} ViewManager;
```

parch 表示并行系统结构信息，其定义如下：

```
typedef struct {
    ArchitectId arch;             /* 系统结构标志 */
    NetworkTopo topo;             /* 网络拓扑 */
```

```
    ProgMode prog;                  /* 并行程序环境 */
} ParallelArch;
```

系统结构标志表示并行系统的体系结构类型，可为 MPP 结构(ARCH_MPP)、NOW 结构(ARCH_NOW)、其它结构(ARCH_ANY)之一。网络拓扑可为总线(TOPO_BUS)、网格(TOPO_MESH)、蝶形(TOPO_BUTTERFLY)、立方体(TOPO_CUBE)、超立方体(TOPO_HYPERCUBE)、环形(TOPO_RING)、星形(TOPO_STAR)等。并行程序环境可为消息传递(MESSAGE_PASSING)、共享变量(SHARED_MEMORY)及其它(OTHER)。

性能数据缓冲区是为了解决因大容量性能数据集而造成内存匮乏设置的。它由视图管理器管理，负责和磁盘空间的交换。其定义如下：

```
typedef struct _PDataQueue {
    PData * data;                   /* 指向某一视图性能数据缓冲区 */
    struct _PDataQueue * next;      /* 链表指针 */
} * PDataQueue;
```

视图播放控制面板结构用于存储动态视图播放时的控制信息。当屏幕上打开多个动态视图时，可在视图管理器的统一控制下同步播放。其结构定义如下：

```
typedef struct _PlayPanel {
    int status;                     /* 当前播放状态 */
    int nviews;                     /* 播放时需刷新的动态视图个数 */
    int speed_ratio;                /* 相对播放速率 */
    long r_time0;                   /* 视图最小刷新时间 */
    long r_time1;                   /* 实际视图刷新周期 */
    long min_e_step;                /* 最小事件步间隔时间 */
    long time_incr;                 /* 刷新步长 */
    long time_elapsed;              /* 播放时间累计 */
} PlayPanel;
```

当前视图播放状态可为以下值之一：

PLAY_START	播放复位状态，即视图还未开始播放
PLAY_PLAYING	播放状态，即“Play”按钮按下后的状态
PLAY_PAUSE	暂停状态，即“Pause”按钮按下后的状态
PLAY_STEP	单步播放状态，即“Step”按钮按下后的状态
PLAY_END	播放完毕状态，即视图已播放到末尾
PLAY_FF	快进状态，即“FastForward”按钮按下后的状态

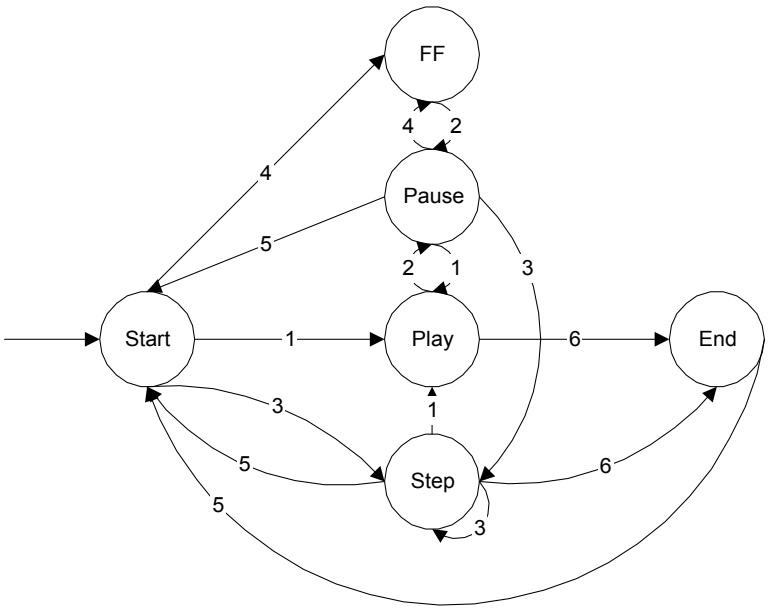
视图最小刷新时间是各个打开的动态视图的刷新时间之和。该值决定了系统视图播放的最快速度，即视图管理器最快只能以 r_time0 作为刷新周期。播放时用户可以设置一个速度缩放系数 speed_ratio，该值在 1 到 100 内可变。当 speed_ratio 为最大值 100 时，实际视图刷新帧周期为 r_time0，对应于最快播放速度；当 speed_ratio 为最小值 1 时，实际视图刷新帧周期为 r_time0 的 100 倍，对应于最慢播放速度。刷新步长 time_incr 为一次刷新操作对应的应用程序实际执行时间。播放时间累计 time_elapsed 为当前对应的应用程序实

际执行时间的累计值。

系统中所有视图创建时都需向视图管理器注册，加入由视图管理器维护的视图链表中，视图结构的指针成员 views 为链表首指针。对所有视图的操作请求都将传递给视图管理器，视图管理器在视图链表中查找对应的视图指针，然后调用该视图结构的相应操作接口函数。

4.3.2.2 视图播放控制与视图播放状态转换

动态视图在播放中可处于不同的播放状态，分别对应于不同的播放按钮按下的状态。视图的播放状态转换图如图 14所示：



①Play 按钮按下	Start	视图播放初始状态
②Pause 按钮按下	Playing	视图正常播放状态
③Step 按钮按下	Pause	视图播放暂停状态
④FastForward 按钮按下	Step	视图单步播放状态
⑤Rewind 按钮按下	FF	视图快速播放状态
⑥视图播放到末尾	End	视图播放到末尾状态

图14 视图的播放状态转换图

4.3.2.3 视图相关性控制

视图间的相关性是由视图管理器进行管理和控制的。视图相关性控制包括视图的同步控制和视图间的关联控制。在视图的播放过程中，各个视图所表现的数据应是同步的，即各个视图使用相同的性能数据集，同时，动态类视图的播放应保证时间一致性。视图的同步控制就是对视图进行同步控制的一种机制。视图间的关联控制则是指视图与源代码间的关联以及动态类视图间的时间关联，即根据一个视图指定图形对象所代表的性能信息(包

括源代码信息、时间信息及其它相关信息)与其它视图(包括源代码)对应图形对象相互参照定位的功能。视图间的关联有助于视图的相互参照对比。

由于每一个打开的视图都注册在视图管理器中，因此视图管理器通过同步调用视图的播放控制接口达到对视图的同步控制功能，通过调用动态类视图的时间定位控制接口达到视图间的关联控制。

4.3.3 视图

4.3.3.1 视图类型

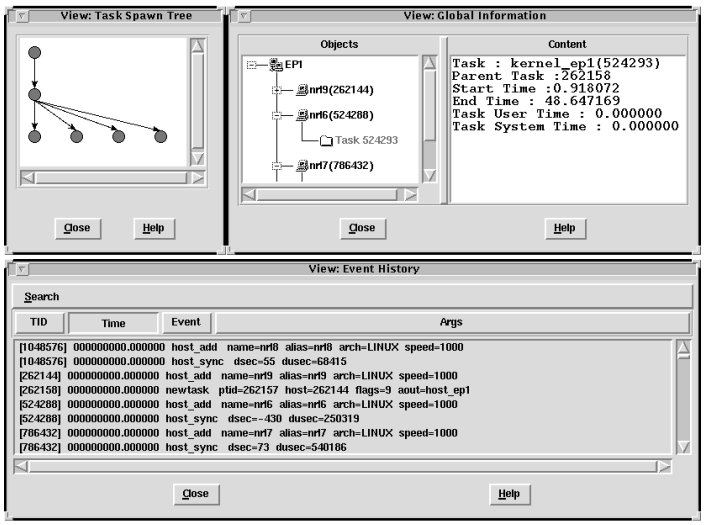


图15 系统信息类视图

- **系统信息类视图**

系统信息类视图均为静态类视图，用于表现系统的静态信息。这类视图包括任务生成树(Task Spawn Tree)、全局信息(Global Information)和事件记录(Event History)。

任务生成树(图 15上左)以树的形式通过从父任务到子任务间的连线描述了各个并行任务间的关系。全局信息(图 15上右)以层次显示的形式给出了并行应用程序、主机及并行任务三个层次的关系及每一层次上相应的全局信息。事件记录(图 15下)是按照SDDF 文件的含义将 Trace File 中的信息直接表示出来，这对一些熟练的程序员进行程序调试、分析有很大帮助。

- **利用率类视图**

利用率类视图是用于表现并行系统在执行应用程序过程中的效率的一类视图。这类视图包括利用率统计图(Utilization Summary)、利用率计数图(Utilization Count)、利用率曲线(Utilization Plot)、利用率 Kiviat 图(Utilization Kiviat)和并发性剖析图(Concurrency Profile)。

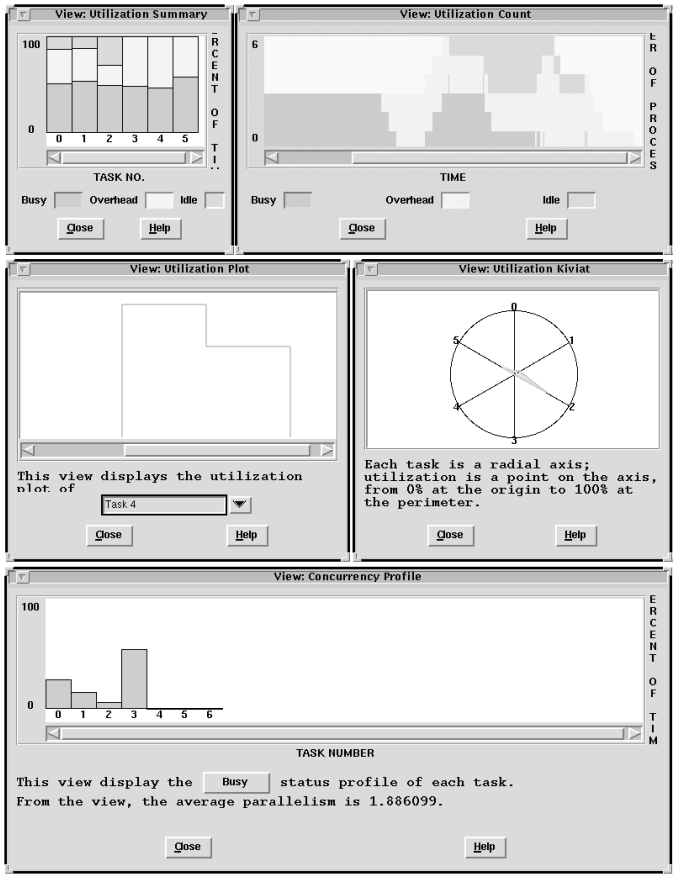


图16 利用率类视图

利用率统计图(图 16上左)显示了每一个任务中三种状态(忙、空闲和开销)所占的比例。图中以横轴表示任务，以纵轴表示比例，是一个静态视图。利用率计数图(图 16上右)以时间为横轴，任务数为纵轴，显示每一时刻处于忙、空闲和开销三种状态的任务数。利用率曲线(图 16中左)与利用率计数图类似，也是以时序的方式描述每一时刻处理机状态的视图，不同之处是利用率曲线仅显示处于忙状态任务数随时间变化的情况。利用率 Kiviat 图(图 16中右)是用来描述任务负载平衡的动画类视图。在这个视图中，表示任务的结点均匀地分布在一个圆周上，由圆心到该点的半径表示在一定时间区间内忙状态所占的比例：圆心处为 0，圆周上为 1。从将各任务忙状态比例对应点连接成的多边形可以看出各任务负载平衡情况：低的负载将使多边形的顶点靠近圆心，高的负载将使多边形的顶点靠近圆周。并行性剖析图(图 16下)表现了处于某一状态有特定数目任务的时间所占的百分比。纵轴表示时间的百分比，横轴表示任务数目，所要统计的状态在视图的下方进行设定。这个视图是一个静态视图。

- **通信类视图**

通信类视图用于表示系统通信情况，这类视图包括通信网络图(Communication Network)、通信统计图(Communication Summary)、通信动画(Communication Animation)、通信矩阵(Communication Matrix)、通信流量图(Communication Traffic)、消息队列(Message Queue)和时空图(Space-time)。

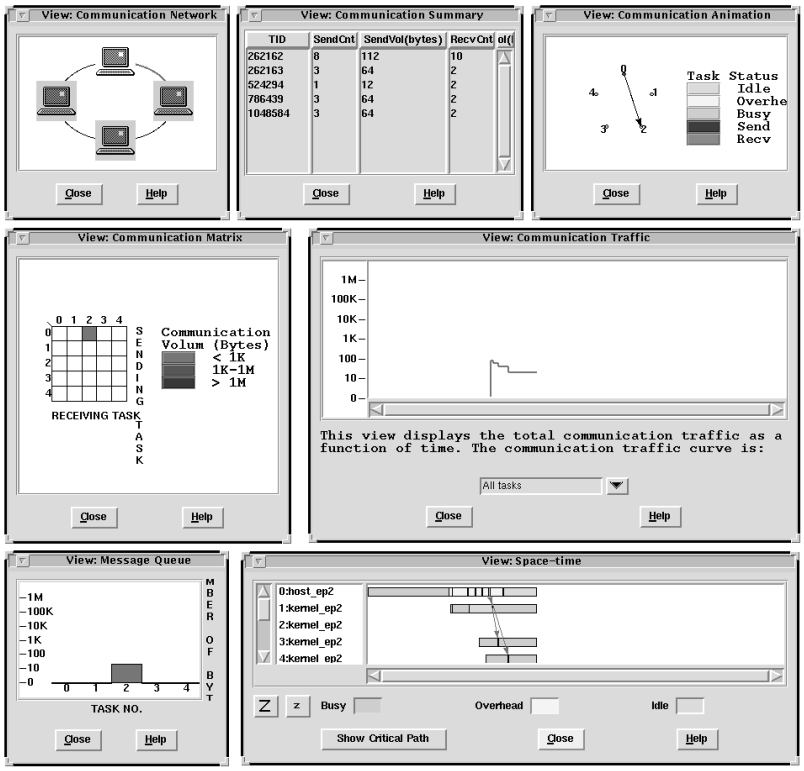


图17 通信类视图

通信网络图(图17上左)以动画的方式将结点间的通信过程映射到网络拓扑图上，VENUS 系统提供了总线型网络、环网、星形网络、立方体网络及二维网格等多种网络拓扑。通信统计图(图 17上中)以表格的形式表示每个任务收、发消息的次数、长度，以及因接收消息而造成的阻塞时间等。通信动画(图 17上右)与通信网络图类似，用动画的方式表示任务间的通信情况，只是它并不将通信映射到网络拓扑之上。通信矩阵用矩阵(图 17中左)的形式描述通信情况，矩阵中每一个小方格(坐标为 i，j)表示由任务 i 向任务 j 发送消息，小方格的颜色反映出消息的长度。通信流量图(图 17中右)以时间为横轴描述了网络中每一时刻消息的总量，它体现了网络的拥挤程度。消息队列(图 17下左)以动画的形式表示出处于消息队列中的消息长度的变化情况。由于 PVM 采用异步消息发送机制，因此可以认为没有发送消息队列。在这个视图中表现的是接收方的消息队列。时空图是性能可视化中常见的一种视图，也是包含信息量最多的视图之一。时空图(图 17下右)以时间为横轴，在图中表示出每一任务状态变化(以矩形表示)，同时对于每一次消息传递，以一个有向线段从发送方指向接收方表示。

- **并行任务类视图**

并行任务类视图是用于表现有关并行任务的特定信息的视图。这类视图主要包括并行任务 Gantt 图(Task Gantt)和并行任务统计图(Task Summary)。

并行任务 Gantt 图(图 18上)以时间为横轴，描述了用户指定对象(函数或代码段)在任务与时间构成的空间中的分布情况。并行任务统计图(图 18下)与利用率统计图类似，与之不同的是并行任务统计图统计的对象是用户所关心的函数或代码段。

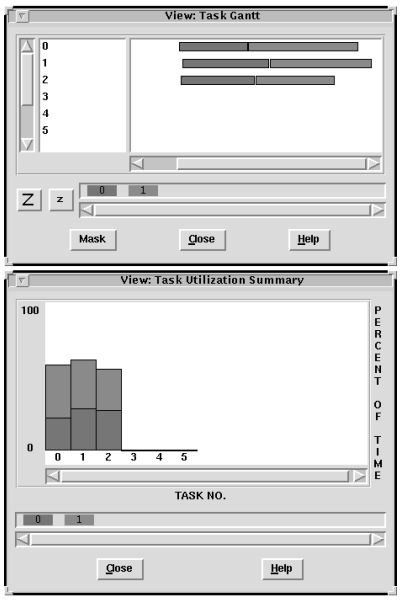


图18 并行任务类视图

4.3.3.2 统一的视图接口

与视图模型对应，定义如下的统一视图结构：

```
typedef unsigned long ViewId;
typedef unsigned long PresentationId;
typedef unsigned long Dimension;
typedef unsigned long VisualType;
typedef unsigned long CoordOrient;
typedef unsigned long ZoomLevel;
typedef unsigned long SpeedRatio;
typedef unsigned long Color;
typedef char * String;

typedef void * PData;
typedef void (*Vfp)();
typedef int (*Ifp)();
typedef long (*Lfp)();

typedef struct {
    int npatterns;              /* 感兴趣的性能事件数 */
    EMask * patterns;           /* 感兴趣的性能事件集合 */
} Protocol;                     /* 性能数据子集 */
typedef struct {
    PresentationId p_id;        /* 视图表现形式 */
    Dimension dim;              /* 视图维数 */
    VisualType v_type;          /* 视图类型 */
    CoordOrient c_type;         /* 坐标系方向 */
    ZoomLevel scale;            /* 视图细节度 */
    Color fg;                   /* 视图缺省前景色 */
    Color bg;                   /* 视图缺省背景色 */
    char show_grid;             /* 是否显示网格 */
```

```
    char show_rulertag;          /* 是否显示刻度 */
} VVisual;                       /* 视图的可视化属性 */
typedef struct _View {
    ViewId id;                   /* 视图号 */
    String description;          /* 视图说明信息 */
    Protocol protocol;           /* 视图性能数据协议 */
    VVisual visual;              /* 视图可视化抽象 */
    PData data;                  /* 视图性能数据缓冲区 */
    char status;                 /* 视图状态 */
    long r_time;                 /* 视图刷新时间 */
    Ifp initialise;              /* 视图构造接口 */
    Ifp filter;                  /* 视图过滤接口 */
    Ifp display;                 /* 视图显示接口 */
    Ifp control;                 /* 视图属性控制接口 */
    Ifp analyse;                 /* 视图性能分析接口 */
    Ifp destroy;                 /* 视图析构接口 */
    Ifp refresh;                 /* 视图刷新接口 */
    Ifp blank;                   /* 视图复位接口 */
    Ifp zoom;                    /* 视图缩放接口 */
    Ifp timelocate;              /* 视图时间定位接口 */
    Lfp test;                    /* 视图刷新时间测试接口 */
    struct _View * next;         /* 视图接口 */
} View , *ViewList;              /* 视图 */
```

性能数据协议结构与视图模型的性能数据分析抽取层对应。视图结构的 protocol 成员描述了本视图所关注的性能事件的集合(性能事件用其事件号标识)。亦即在建立自己的局部性能数据集时，视图根据 protocol 成员确定性能事件的抽取匹配模式。

视图可视化抽象成员 visual 与视图模型的性能数据可视化表示层对应，该结构从以下几个方面描述了视图可视化表示的抽象表示：

- **视图表现形式(p_id)**

可为文本描述(TEXT)、直方图(BAR)、饼图(PIE)、曲线图(CURVE)、网络图(NET)、树形图(TREE)、图象(IMAGE)、音频(AUDIO)、动画(ANIMATION)等。

- **视图维数(dim)**

可为二维视图或三维视图。

- **视图类型(v_type)**

可为基于统计分析的静态类视图(VISUAL_STATISTIC)、动画类视图(VISUAL_ANIMATION)、历史记录类视图(VISUAL_HISTOGRAM)和其它类型的视图(VISUAL_CUSTOM)。

- **视图采用的坐标系方向(c_type)**

坐标系原点可在视图左上角(COORD_UPPERLEFT)或左下角(COORD_BOTTOMLEFT)。

- **视图细节度(scale)**

  表示视图放大缩小的程度。
- **视图缺省前景和背景色(fg、bg)**
- **视图中是否显示网格(show_grid)**
- **视图中是否显示坐标刻度(show_rulertag)**

视图结构成员 data 是一个 void 类型指针，该指针指向视图的局部性能数据集。在具体实现时，每一视图可以定义自己的局部性能数据集的结构，并通过强制类型转换存取 data 成员。视图状态 status 可为 VIEW_ON 或 VIEW_OFF，表示视图是否已打开。视图刷新时间 r_time 为动态视图刷新一帧所需时间，该时间将由系统通过实测获得。

系统为每一视图对象定义了一个全局的视图结构指针。

4.3.3.3 细节度动态自动调节技术

细节度动态自动调节技术是 VENUS 系统可视化模块采用的一种动态视图播放技术。细节度是指视图显示的细致程度。在可视化的过程中，所采用的性能数据越多，则越能够精确地表现应用程序的性能，其细节度越高。然而，随着数据量的增大，视图的播放速度会下降。为了达到最佳的效果，就需要在播放细节度与播放速度之间进行权衡。而这种权衡的标准又是一般用户难以掌握的。为此，VENUS 系统中采用了细节度动态自动调节的技术，在可视化数据显示层中根据物理设备的精度，将视图在当前显示比例下不可见的细节忽略，而当改变显示比例时修改已显示的视图。

在 VENUS 系统中，对动态类视图定义了一个内部函数 Detailization()用以完成细节度的动态自动调节。在每个视图的播放过程中，通过对显示设备的检测和视图播放比例的设置，自动地将显示设备无法表现的性能数据忽略，从而在不影响视图性能的情况下提高视图的播放速度。在对视图进行缩放控制时，通过调用 Detailization()函数将视图在放大后能够表现出来的细节重新显示在显示设备之上，从而保证了视图放大后对性能数据的表现能力。

4.3.3.4 坐标变换与视图内容的自动布局控制

当视图窗口的大小变化时，视图窗口内的视图大小也应随之变化，视图窗口内容的布局控制是用来控制视图窗口大小变化后视图的刷新显示。

由视图窗口大小变化引起的视图变化包括视图图形的位置变化(平移)和视图图形的大小变化(缩放)。为了处理的方便，视图不采用系统窗口的绝对坐标系，视图的大小位置变化也以参照坐标系给出。

由于窗口系统的图形是以绝对坐标进行控制的，而视图的刷新控制是以参照坐标系给出的，因此在这里需要进行坐标转换。设*(x, y)*表示变换前坐标，*(X, Y)*表示变换后坐标，*(x', y')*表示变换前绝对坐标，*(X', Y')*表示变换后绝对坐标，则参照坐标系原点为 *O*，其绝对坐标为(x'_o, y'_o)(变换前)和(X'_o, Y'_o)(变换后)，水平(垂直)缩放比例为 *xScale*(*yScale*)，水平(垂直)平移距离为 *xIncrement*(*yIncrement*)，则

$$\begin{cases} X = x * xSacle + xIncrement \\ Y = y * yScale + yIncrement \end{cases}$$

又，

$$\begin{cases} X' = X + X'_o \\ Y' = Y + Y'_o \end{cases} \qquad \begin{cases} x' = x + x'_o \\ y' = y + y'_o \end{cases}$$

故，

$$\begin{cases} X' = x' * xScale + (xIncrement - x'_o * xScale + X'_o) \\ Y' = y' * yScale + (yIncrement - y'_o * yScale + Y'_o) \end{cases}$$

令，

$$\begin{cases} xIncrement' = xIncrement - x'_o * xScale + X'_o \\ yIncrement' = yIncrement - y'_o * yScale + Y'_o \end{cases}$$

有，

$$\begin{cases} X' = x' * xSacle + xIncrement' \\ Y' = y' * yScale + yIncrement' \end{cases}$$

根据上述公式，可以将按参照坐标系给出的视图刷新控制变换为绝对坐标系下的视图刷新控制。

4.4 辅助模块的实现

辅助模块包括主界面模块、工程管理模块、系统配置管理模块、窗口管理模块、Help模块和辅助工具模块。这些模块的主要功能是完成系统界面，为其它各个模块提供一个统一的界面。这一部分用 Tcl/Tk 语言设计实现。

4.4.1 主界面模块

VENUS 系统主界面模块是创建系统主界面的基本模块，它首先装入系统中的其它模块，并读取系统的配置文件，由此创建系统界面。系统主界面由系统菜单、系统工具栏、视图工具栏、播放工具栏、工程文件行、启用区、状态行和进度指示器组成(如图 19所示)。

4.4.2 工程管理模块

工程管理模块完成工程的管理功能。它包括建立、打开、保存工程文件，对工程辅助信息的维护和工程所包含的项目信息的维护功能。工程管理模块提供了如下函数接口：

```
ProjectNew {}                    /* 建立一个新工程文件 */
ProjectOpen {{name {}}}          /* 打开一个工程文件 */
ProjectSave {}                   /* 工程文件的保存 */
```

```
ProjectSaveAs {}                    /* 工程文件的另存 */
ProjectInfo {}                      /* 工程文件辅助信息维护 */
ProjectWin {{Title "(None)"}}       /* 工程文件项目信息维护 */
```

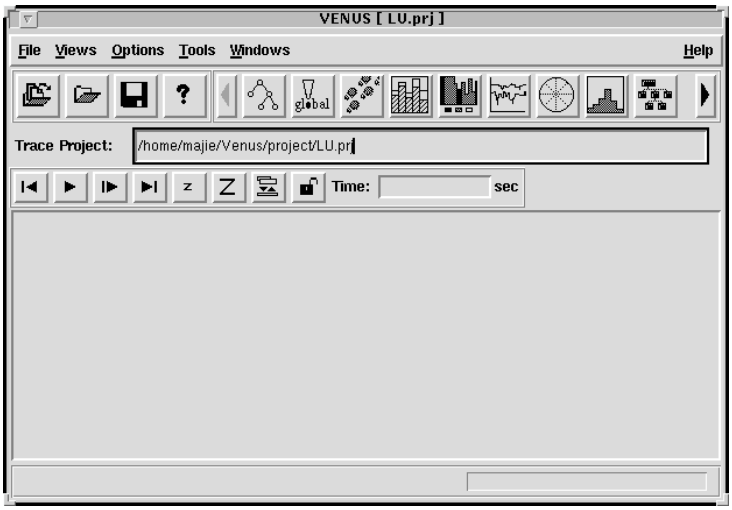


图19 VENUS 系统主界面

4.4.3 系统配置管理模块

系统配置管理模块使用交互的方式由用户设置系统的配置情况。系统配置包括一般属性配置、结构性属性配置和视图属性配置。VENUS 系统通过对话窗口的形式设定各项系统配置。系统配置管理模块提供了如下函数接口：

```
OptionGeneral {}                    /* 一般属性配置管理 */
OptionNetwork {}                    /* 结构性属性配置管理 */
OptionView {}                       /* 视图属性配置管理 */
```

4.4.4 窗口管理模块

窗口管理模块用于管理系统所打开的视图窗口。为了便于用户使用，VENUS 系统将用户所打开的视图窗口排列在系统窗口菜单上，并提供视图窗口列表。窗口管理模块提供了如下函数接口：

```
AddWindow {name prompt}             /* 增加系统窗口菜单项目 */
DeleteWindow {name}                 /* 去消系统窗口菜单项目 */
SeekWindow {title}                  /* 按窗口标题查找窗口 */
WindowCascade {}                    /* 层叠窗口 */
WindowCloseAll {}                   /* 关闭所有窗口 */
WindowList {}                       /* 窗口列表 */
WindowListActivate {parent list}    /* 激活一个已打开的窗口 */
WindowTile {}                       /* 平铺窗口 */
```

4.4.5 Help 系统

Help 系统包括了状态行(Status Line)、按钮提示(Help Tips)、信息提示框(Help Box)和帮助子系统(Help)。VENUS 系统中设置了相应的函数接口来注册各类帮助信息。

```
AddHelpBox {widget message}          /* 设定信息提示框信息 */
AddMessage {widget message}          /* 设定状态行信息 */
AddTips {widget tips}                /* 设定按钮提示信息 */
DefineHelpBox {}                     /* 初始化信息提示框 */
DefineHelpTips {}                    /* 初始化按钮提示 */
DefineHelpMessage {}                 /* 初始化状态行 */
RegisterHotHelp {widget helpType}    /* 设定系统帮助 */
UnRegisterHotHelp {widget helpType}  /* 取消系统帮助 */
```

在 Help 子系统中，定义了一个超文本浏览器(winhelp.tcl)，它能够浏览由 HTML 语言的一个子集所书写的帮助文件，并能够通过文件的链接迅速地获取所需的帮助信息。

4.4.6 辅助工具模块

辅助工具模块定义了一些用于界面设计的构件和其它一些系统实用工具。utility.tcl 模块定义了 VENUS 系统中所使用的实用工具；其它的辅助模块定义了用于构造系统界面所用到的复合型的构件。

```
utility.tcl                          /* 系统实用工具 */
frame.tcl                            /* 框架类构件 */
spinner.tcl                          /* 微调构件 */
animate.tcl                          /* 动画构件 */
combolist.tcl                        /* 下拉型列表框类构件 */
files.tcl                            /* 文件选择框构件 */
xlist.tcl                            /* 特殊列表框类构件 */
tree.tcl                             /* 树形结构图构件 */
```

第五章 应用实例

5.1 概述

本章将 VENUS 系统运用于考察 NAS Parallel Benchmark(NPB)的 PVM 版本的性能。EP Benchmark 是其中的一个核心程序。EP 执行 2^{28} 次循环，在每一循环步内产生一对随机数，并判断其是否能产生高斯偏差[20]。EP 的 PVM 并行版本的构造比较直接，采用 Master/Slave 的程序模式。Master 任务将 2^{28} 次循环分割为若干各独立的部分，将每一部分分配给一个 Slave，各 Slave 计算完后将结果发送回 Master。我们在由四台 PC Linux(P5/133)构成的四个结点的 NOW 环境下对 EP 进行了测试，以建立同构计算环境下各结点计算性能的参考，并使用 VENUS 系统对 EP 程序进行了可视化分析。

5.2 运行环境

EP 的运行环境包括由四台 PC Linux(P5/133)构成的局域网络及运行与 Linux 下的 PVM 软件系统。

PC 的硬件配置为 Pentium 586 处理器，32M 主存储器；网络为 10M Ethernet；Linux 是由网络安装的 Slackware 3.1 版本；PVM 版本为 3.3.11。

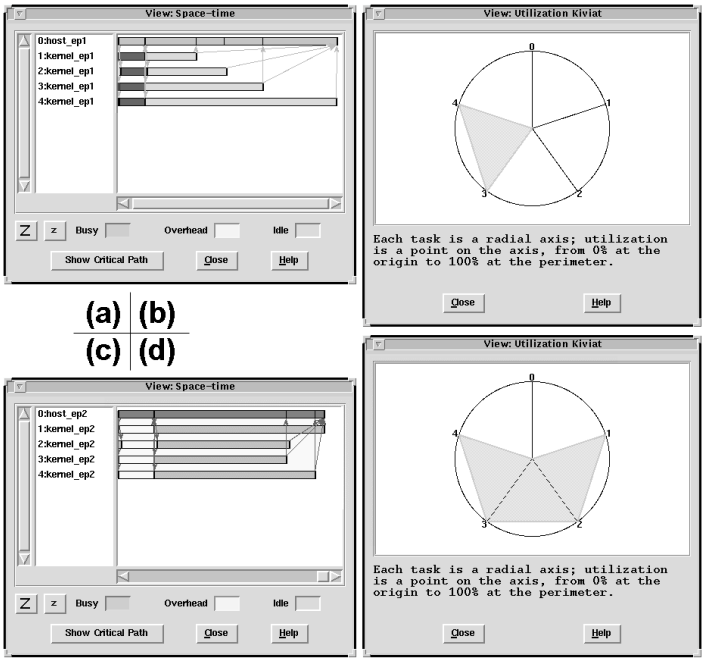
5.3 性能分析

EP1 采用负载随机分配的方式，即 Master 为每一 Slave 随机分配大小不等的循环步数。图 20的(a)和(b)分别为 EP1 的任务时空视图和利用率 Kiviat 图。由(a)可知，由于 Master(任务 0)必须等所有子任务将结果返回才能结束，因此执行速度最慢的子任务就成了制约整个程序性能的瓶颈。由于是随机的负载分配，负载较轻的结点上的子任务很快完成其计算后便退出了，而负载较重的结点上的子任务使整个应用程序的执行时间增长，是系统的瓶颈。从(b)中也可看出在程序的执行后期，处理机 1 和 2 已处于空闲状态。为此，我们对 EP1 进行改进以使其适应同构环境下的计算。在 EP2 中 Master 采用均匀负载分配方式，让各个计算结点承担相同的负载。改进后的相应视图如(c)和(d)所示。

从表 1所示的两个版本的 EP Benchmark 的执行时间也可看出，改进后的 EP2 的各子任务基本同时结束，而整个 Benchmark 的执行时间比 EP1 有了较大下降。

测试程序	程序运行时间(秒)	每一任务运行时间(秒)
EP1	83.217174	19.209/29.855/44.771/72.905
EP2	58.401264	37.304/37.319/45.551/48.059

表1 EP Benchmark 的两个版本的性能比较



(a) EP1 任务时空视图 (b) EP1 利用率 Kiviat 视图
(c) EP2 任务时空视图 (d) EP2 利用率 Kiviat 视图

图20 EP1 和 EP2 的性能视图

第六章 VENUS 系统性能分析

6.1 VENUS 系统对并行程序性能优化的效果

VENUS 系统是用于并行应用程序性能优化分析的软件系统，它通过对并行应用程序的性能数据进行分析得出应用程序的性能特征，对应用程序的性能给予评价并给出优化建议。为了对 VENUS 系统的性能进行分析，这里对一组典型并行应用程序进行测试。通过对这些典型应用运行性能的实测、分析与优化，对用 VENUS 系统进行性能分析优化的效果进行评价。

并行应用程序从其结构上说，可以分为定义域分解类的并行应用程序、控制分解类的并行应用程序以及包含不同结构的异构类并行应用程序。如果一个基于大型静态数据结构的计算对每个数据元素的工作量大致相同，那么将问题按照其定义域进行均匀划分的方法称为定义域分解技术，采用这种技术生成的并行应用程序称为定义域分解类的并行应用程序。当定义域和数据结构都不太规则或不可预测时，常用的方法是去分布计算的控制流而不是去分布定义域。这种方法称为控制分解技术，它包括功能分解法和称为“经理-工人”的方法。功能分解法是将不同的功能划分为不同的功能模块，并分布在不同的处理机结点上；“经理-工人”方法是一种分治(divide-and-conquer)技术，其思想是把应用问题分解成大小不一定相同的任务，并让一个进程作为经理(Master)来进行管理，而将其余的结点作为工人(Slave)结点，完成经理所分配的任务。对于一个较为复杂的问题，很难单纯用一种方式进行问题的分解，在解决这类问题的过程中，我们经常使用的程序分层法在不同的层次上使用不同的分解技术来发掘并行性。采用这种方法所形成的并行程序称为异构类并行应用程序[13]。

在这里使用了并行遗传算法(PGA)解决旅行商问题(TSP)、快速排序问题、线性插值问题、矩阵的 LU 分解问题、高斯消去法解线性方程组问题及利用克莱姆法则解线性方程组问题等应用程序作为性能测试的基准程序，对 VENUS 系统对并行程序性能优化的效果进行了评价。表 2列出了 VENUS 系统在清华工作站机群上测试的结果。通过上述对并行应用程序性能优化的测试可以得出 VENUS 系统能够有效地对并行应用程序进行分析优化。

6.2 性能数据的表示形式与对外部存储器的要求

在 VENUS 系统中，性能数据集是以 SDDF 格式的文本文件方式存储于外部存储器中的。随着被测程序规模的扩大，由 VENUS 系统形成的性能数据文件也随之增大。因此，VENUS 系统在进行性能可视化分析优化的过程中对系统的外部存储器有一定的要求。

针对上述实例，对性能数据文件的大小进行了实际测量，表 3列出了 VENUS 系统在清华工作站机群上测试所得的性能数据文件的大小与所记录的事件数目的比较。在进行测试时，性能监测部分收集了并行应用程序的全部性能数据。因此，在针对某一特定性能问

题进行分析时，性能数据文件的长度还可以进一步减小，从而使 VENUS 系统对外部存储器的要求降低。根据对 SDDF 文件格式的分析不难得出平均记录一个事件所需的外部存储器容量约在 70 至 100 字节之间。

应用程序	执行时间	分析结果	优化提示	优化程序执行时间
并行遗传算法(PGA)解决旅行商问题(TSP)	120.338130	同步相持时间过长	调整计算与通信的位置	77.438888
快速排序问题	12.239837	消息通信过多	减少消息通信次数	8.985184
线性插值问题	2.292626	消息通信过多	减少消息通信次数	0.798944
矩阵的 LU 分解问题	19.009233	并行任务粒度过细	增大任务粒度	1.648623
高斯消去法解线性方程组	22.156484	并行任务粒度过细，通信次数过多	增大任务粒度	2.965799
利用克莱姆法则解线性方程组	2.435610	未检测到严重的性能瓶颈	——	——

表2 VENUS 系统对并行应用程序性能优化的测试结果

应用程序	性能数据文件长度(Bytes)	所记录的事件数目	平均每个事件的长度(Bytes)
并行遗传算法(PGA)解决旅行商问题(TSP)	50880	631	80
快速排序问题	145195	2165	67
线性插值问题	57732	737	78
矩阵的 LU 分解问题	31565	453	70
高斯消去法解线性方程组	232788	3194	73
利用克莱姆法则解线性方程组	14444	133	109

表3 性能数据文件的容量

6.3 性能数据的内部表示与对主存储器的要求

在 VENUS 系统进行性能分析之前首先要将性能数据读入主存储器，因此，主存储器的容量将直接限制了 VENUS 系统所分析的应用程序的规模。

根据 VENUS 系统的设计与实现(参见第四章)不难得出，对于 VENUS 系统性能数据集中的每一组事件记录(相互匹配的二个事件记录)将产生一个状态数据(State)，同时，对于每一对消息发送与接收(pvm_send 和 pvm_recv)将产生一个消息数据(Message)。假设每个整型数据长度为 4 字节，则一个 State 结构需要 56 字节，一个 Message 结构需要 72 字节。对于有 N 个事件的一个性能数据集，若其中消息事件所占比例为 p，则其所需的主存储器空间为(28+18p)N，这里我们认为每四个消息事件(pvm_send0、pvm_send1、pvm_recv0 及 pvm_recv1)构成一个完整的 PVM 消息，并忽略了 Host、Task 等其它内部数据结构。这样，对于一个具有 1M 个事件记录，其中消息事件占 60%的性能数据集，其所需的主存储器空间为 38.8M。

6.4 可视化视图的显示速度

显示速度是一个可视化系统的重要性能之一，VENUS 系统中采用了细节度动态自动调节技术提高了视图的播放速度。与一般的视图播放技术相比，细节度自动调节技术能够有效地改善动态类视图的播放速度。表 4列出了应用程序时空图在 Sun Ultra 1 上两种不同播放技术的播放性能比较。

应用程序	实际时间(秒)	播放时间(秒)	
		一般视播放技术	细节度动态自动调节技术
并行遗传算法(PGA)解决旅行商问题(TSP)*	77.438888	87	86
快速排序问题	8.985184	26	10
线性插值问题	0.798944	6	3
矩阵 LU 分解问题	1.648623	53	12
高斯消去法解线性方程组	2.965799	84	18
利用克莱姆法则解线性方程组	2.435610	56	7

表4 动态类视图播放技术性能比较

通过调节视图的缩放比例，还可以进一步提高视图的播放速度，使之达到与实现执行时间近似相等的程度。

* 由于 TSP 问题通信事件较少，因此其性能数据细节度较粗，采用细节度自动调节技术对它的播放效果没有很大的影响。

第七章 结束语

7.1 结论

本文在结合 863 项目**大规模并行处理体系结构的关键技术**的研究过程中，针对基于消息传递机制的并行系统性能进行可视化及分析的研究。从理论上建立可视化的基本模型，并根据这一模型实现了 VENUS 系统。VENUS 系统现以完成如下两个版本：

1) 应用于曙光计算机(MPP)结构的版本；

2) 应用于可扩展机群硬件平台的版本；

此外，还在 IBM RS/6000 以及由 Sun SPARC 1+、Sun Ultra1、Sun Ultra2 和 PC Linux 构成的异构 NOW 环境等多种平台下进行了移植。

VENUS 系统提供在曙光计算机(MPP 结构)和可扩展机群两个硬件平台上实用的并行程序行为监测与性能辅助分析软件系统。这一软件系统能够在无需用户修改应用程序的情况下获取低失真率和低噪音的性能数据，并按照数据抽象方法进行组织，以可视化方式直观表达监测和分析的过程与结果。利用该工具对典型应用能够准确反映并行程序运行过程中存在的性能瓶颈，并以与用户交互的方式对程序优化给出提示。

VENUS 系统达到了 863 项目合同所要求的指标，对于典型问题的平均测量代码插入率不超过源代码的 20%，对于典型问题性能瓶颈定位的准确率平均达到 80%以上。同时用户界面友好，对有关的性能瓶颈问题均能以可视化方式直观表达，对程序优化的提示明确有效。

7.2 进一步的工作

- **性能数据可视化表现形式的进一步研究**

性能数据可视化的关键在于如何能有效地用图形来表现性能数据。在 VENUS 系统中我们提供了包括时空图、Kiviat 图、通信矩阵等在内的十七种视图。但是，如何使视图更加直观更有利于性能分析仍旧是可视化研究的一个关键领域，它对并行应用程序性能分析将起到重要作用。

- **系统性能的改善**

VENUS 系统性能主要体现在它的视图对并行应用程序性能的表现能力和性能分析的有效性方面。通过对可视化表现形式的进一步研究能够提高视图对并行应用程序性能的表现能力，而性能分析的有效性则需通过对并行系统硬件的深入了解和对并行性理论的进一步研究来提高。在 VENUS 系统中，我们所采用的性能分析模型是基于假设验证的性能模型，因此，假设的提出和验证的标准的确定是性能分析有效与否的关键，在这一方面的研究将对 VENUS 系统性能的改善起到重要作用。

- **使用方便性的提高**

VENUS 系统是一个实用的应用程序开发工具，它面向广大的程序员。而这些从事并行程序开发的程序员可能是其它科研领域的研究人员，这就要求 VENUS 系统是个易学易用的系统。因此，提高 VENUS 系统的可用性是 VENUS 系统推广应用所必须的工作。

致 谢

通过三年的研究生学习和二年来课题的研究工作，使我在科研工作方面有了进一步的实践和锻炼。在此，首先感谢我的导师郑守淇教授，感谢他在三年来对我的悉心指导，以及在课题和论文工作中对我的指导、启发和建议。郑老师渊博的学识、严谨的治学态度和创新精神，使我三年来的学习和科研工作收益匪浅。在此向郑老师表示衷心的感谢。

这里，我还要感谢 863 课题的主要负责人赵银亮副教授和课题的主要设计人钱德沛教授，感谢他们在课题研究中对我的指导和帮助。此外还要感谢石晓虹博士生在可视化系统设计及实现过程中对我的帮助，以及课题组其它成员在课题研究过程中对我的帮助。他们对本论文的完成给予了许多有益的建议和帮助。

在西安交通大学数年的求学过程中，我还得到了其它许多老师、同学的关心和帮助，在此对他们一并表示感谢。最后，也感谢我的家人这些年来对我的关心和鼓励，特别感谢他们在我论文工作中给予的支持和帮助。

参考文献

[1] E.Kraemer, J.Stasko. “The Visualization of Parallel System: an Overview”. J. Parallel and Distributed Computing, Vol.28, No.11, pp.30~36

[2] Michael T. Heath (University of Illinois)， Jennifer A. Etheridge (Oak Ridge National Laboratory)， “Visualizing the Performance of Parallel Programs”， IEEE Software，September 1991， pages 29~39.

[3] Barton P. Miller，Jeffrey K. Hollingsworth，Mark D. Callaghan，“The Paradyn Parallel Performance Tools and PVM”

[4] D. Kranzlmuller， R. Koppler， S. Grabner， Ch. Holzner， J. Volkert， “Parallel Program Visualization with MUCH”.

[5] 清华大学计算机系，“可扩展并行机群系统可视化人机交互并行程序集成开发环境 IPCE 使用说明”，1996 年 3 月。

[6] Adam Beguelin and Vaidy Sunderam， “Tools for Monitoring， Debugging， and Programming in PVM”， PVM’96， pages 7~13.

[7] 国家智能计算机研究开发中心 张兆庆 刘强，“并行程序行为及性能的可视化工具——ParaVision”，软件世界 1996 年 5 月 pages 86~87

[8] Thierry Delaitre， Fancois Spies， Stephen Winter， “Simulation Modeling of Parallel Systems in the EDPEPPS Project”， UK Parallel ’96， pages 1~13.

[9] 中科院计算所 CAD 开放实验室 唐卫清 刘慎权 余盛明 李华，“科学计算可视化”，软件世界 1996 年 5 月 P74~77

[10] V. S. Sunderam, G. A. Geist, J. J. Dongarra, etal. “The PVM Concurrent Computing System: Evolution, Experiences and Trends”. J. Parallel Computing, 1994, Vol.20, No.4, pp.531~546

[11] 沈斌 杨烈文 编译，“并行虚拟机的近期发展”，电子计算机 1996 年第 4 期，pages 32~44

[12] 浙江大学 CAD&CG 国家重点实验室 蒋纯 潘志庚 石教英，“PVM 结构分析(4)——PVM 消息机制”，计算机工程与应用 1996 年 4 月 P69~72

[13] 黄铠，“高性能计算机系统结构：并行性、可扩展性、可编程性”，1995 年，清华大学出版社

[14] Aydt R. “SDDF: the Pablo Self-Describing Data Format”, Technical Report, Urbana-champaign, Ill. Dept. of Computer Science, University of Illinois, 1993

[15] 赵银亮等，VENUS 系统分析与设计报告，1998 年 4 月

[16] 刘坚，基于 PVM 环境的并行程序性能监测的方法与实现，1997 年，西安交通大学硕士学位论文

[17] John K. Ousterhout, “Tcl and the Tk Toolkit”, Addison-Wesley Publishing Company,

1994

[18] 石晓虹等，VENUS 系统性能可视化部分的设计报告，1997 年

[19] 赵银亮等，VENUS 系统性能分析部分的设计报告，1998 年 4 月

[20] S. White, A. Alund and V. S. Sunderam, “Performance of the NAS Parallel Benchmarks on PVM Based Networks”, http://science.nas.nasa.gov/Pubs/TechReports/RNRreports /otherpeople /RNR-94-008/RNR-94-008.html