ISSN 1000-1239/CN 11-1777/TP
48(9): 1693-1703, 2011

MIOS: 面向大规模 CCNUMA 系统的多实例操作系统

卢 凯[1] 迟万庆[1] 高颖慧[2] 冯 华[1]
[1](国防科学技术大学计算机学院 长沙 410073)
[2](国防科学技术大学自动目标识别实验室 长沙 410073)
(kailu@nudt.edu.cn)

MIOS: A Scalable Multi-Instance OS for Large Scale CCNUMA System

Lu Kai[1], Chi Wanqing[1], Gao Yinghui[2], and Feng Hua[1]
[1](*College of Computer, National University of Defense Technology, Changsha* 410073)
[2](*Key Laboratory of Automatic Target Reorganization, National University of Defense Technology, Changsha* 410073)

Abstract MIOS is a scalable operating system designed for large scale CCNUMA system. It introduces multi-instance kernel structure. In MIOS, each instance of OS kernel executes the same code, but runs on a node of the CCNUMA machine and manages its resources respectively. The MIOS provides a single system image running environment for all nodes of CCNUMA system, supporting process and thread task model. Aiming at the features of CCNUMA system and the requirements of scientific computing applications, the MIOS provides several optimizations, including weak shared thread model, cascaded task scheduling, adaptive communication between tasks and register-based lock. We have implemented MIOS on our Galaxy parallel computer system, a large scale CCNUMA system including 2048 processors. The evaluations on Galaxy system, including micro-benchmarks and real parallel applications, show that MIOS can provide comparable performance with a conventional OS for MPI applications. For OMP applications, the MIOS also can provide a good performance speedup on the large scale CCNUMA system with 2048 processors. The structure of MIOS can also provide experiences for designing operating system on many-core processor.

Key words CCNUMA; multi-kernel system; parallel executing model; memory management; operating system

摘 要 MIOS是一个面向大规模CCNUMA系统设计的新型高可扩展操作系统.MIOS创新地采用了多实例内核结构,每个内核实例执行相同代码,分别独立运行和管理一个处理器,多核间通过分布存储管理构成高可扩展的一致性系统映像空间,支持弱共享进程、线程并行模型.MIOS针对大规模CCNUMA系统特点和高性能并行科学计算应用的需求,采用了显式共享数据分布、层次式任务调度、自适应任务间通信以及寄存器锁等优化.在大规模CCNUMA体系结构的银河深度并行计算机上的测试表明,MIOS对MPI应用具有同传统操作系统类似的性能,并可以有效支持2048处理器规模的OMP应用高效运行,具有良好的系统可扩展性.

关键词 CCNUMA;多核系统;并行执行模型;存储管理;操作系统

中图法分类号 TP316

收稿日期:2010-03-22;**修回日期**:2011-03-23
基金项目:霍英东青年教师基金项目(111072)

大规模 CCNUMA(cache coherent non-uniform memory access)系统,指通过缓存(cache)一致性协议构成的紧耦合共享存储的大规模计算机系统,其结构如图 1 所示. 系统中每个处理器都是完整的计算机系统,都带有局部的存储器和外设. CCNUMA 系统具有设备访问延迟不一致的特点,包括内存访问延迟和设备访问延迟等. 并且随着系统规模的扩大,访问延迟的差异性将更加显著. 不一致的访问延迟是 CCNUMA 系统的最大特点和优化重点.

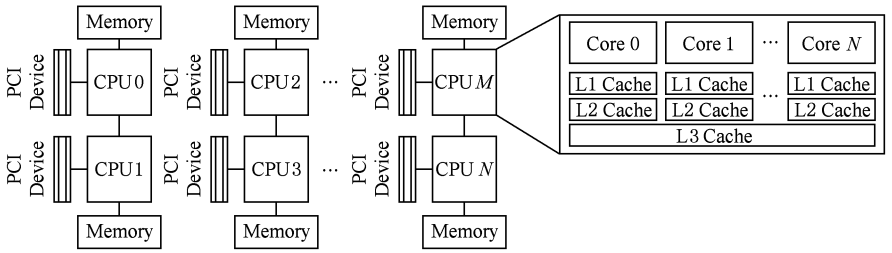


Fig. 1 Architecture of CCNUMA system.

图 1 CCNUMA 体系结构

随着硬件技术的进步,基于多个微处理器构造大规模 CCNUMA 系统将变得越来越方便快捷. Intel 最新的 XEON Nehalem EX 处理器和 Itanium Tukwila 处理器都采用了 QPI 直连技术[8],可以快速构成 64 核的 CCNUMA 系统. 此外,AMD 处理器也提供了 HT[9]互连技术. 可以预计,随着多核处理器技术和互连技术等的进一步发展,会出现包含几千个核的大规模 CCNUMA 系统.

目前的主流操作系统都是由面向单处理器或小规模 SMP 系统的内核架构上发展而来的,如 UNIX,Linux,Windows 和 MINIX 操作系统等. 操作系统结构主要包括宏内核结构(monolithic kernel)、微内核结构(microkernel)和外内核结构(exokernel)等. 目前针对 CCNUMA 系统的操作系统改进,主要是针对访问延迟的差异性,研究如何通过提高数据访问局部性来提升操作系统和应用的运行性能.

SGI 公司针对 Altix 系列 CCNUMA 系统开发了包括不连续内存管理技术[6]、多分配策略优化技术[2]和基于 ACPI 的局部性中断处理技术[3]等,并发行了相应的操作系统补丁. 目前此功能已集成到 Linux 2.6 内核中. Linux 操作系统还针对 CCNUMA 系统设计了多调度域技术[5]和 CPUMEMSET 技术[7]. 操作系统根据 CCNUMA 系统的多核处理器耦合关系定义 Core、Socket 和 Clump 等多个调度域,支持用户应用在一个具有紧耦合度的处理器集合上运行. CPUMEMSET 技术则提供了相应的调度接口,用户可以通过该接口定义应用和 CPU/内存的绑定运行关系. 此外,为了进一步提高数据访问的局部性,Linux 操作系统支持内核代码段和只读数据段的复制技术[6]. 也有一些 CCNUMA 系统基于硬件支持和反向映射技术提供了用户数据的迁移和复制[10]. 但由于实际效果不理想,迁移功能在以后的 NUMA 系统中很少采用. IBM 的 AIX 操作系统等也都设计和实现了类似的局部性优化[1]. 虽然采用了大量局部性优化技术,但是针对大规模 CCNUMA 系统,传统单内核操作系统可扩展性仍无法有效满足大规模应用高效运行的需求. 主要表现在以下方面:

1) 无法高效管理庞大的系统资源. 大规模 CCNUMA 系统的处理器数多达数千,目前的单内核操作系统普遍只能支持 256 处理器规模. 当系统规模庞大时,系统的自身运行开销将占总 CPU 计算能力的 50%以上.

2) 内核数据局部性差. 由于单内核操作系统内核数据是全系统共享的,且由于多存储策略接口只是对用户应用提供,因此内核数据基本采用全系统随意放置的策略,导致内核运行效率低下.

3) 基于共享内存的通信. 在大规模 NUMA 系统中,由于本地和原地访存延迟的差异性,共享内存通信的可扩展性将受全系统 Cache 一致性协议处理性能的制约,无法满足大规模系统中低延迟和高带宽的通信需求.

4) 应用数据局部性优化困难. 应用包括系统级管理应用和用户应用. 虽然操作系统提供了多种数据放置和任务亲和调度支持,但是实际使用中用户难以确定正确的优化方式,效果不理想. 同时,由于历史原因,系统级管理应用缺乏针对 NUMA 系统的优化,全部修改可能性不大.

实际测试表明,当本地和远地访存延迟超过

1:4、系统规模超过 4 处理器时,即使采用了亲和调度、多存储分配策略等优化技术,用户应用的并行性能加速比就开始下降. Wentzlaff 所作的测试也表明,当系统规模超过 16 时,系统开销和锁竞争等将消耗大量的计算能力[13].

在多核或多实例操作系统结构研究方面,微软公司分别提出 Barrelfish[11],Helios[12] 和 Factored OS[13] 操作系统等. Barrelfish 面向异构多核处理器的可扩展性需求采用了异构多内核结构,各个内核间采用显式消息通信,通过状态复制支持进程、线程模型. Helios 和 Factored OS 针对如 CPU/GPU 混合和 CPU/可编程 NIC 混合等异构计算模式,采用主、从内核结构,主内核协同从内核共同完成计算任务.但是上述工作仍无法有效支持大规模 CCNUMA 系统的可扩展性需求.

针对传统操作系统内核结构难以满足大规模 CCNUMA 系统的高可扩展性需求的问题,借鉴 Barrelfish,Helios 和 Factored OS 等多核结构操作系统,本文提出了一种面向大规模 CCNUMA 系统的多实例操作系统架构,并设计和实现了 MIOS (multi-instance operating system)操作系统.在采用 CCNUMA 体系结构的银河深度并行计算机系统上实际测试表明,MIOS 具有良好的可扩展性. MIOS 的主要贡献如下:

① 提出了面向大规模 CCNUMA 系统的高可扩展的多实例操作系统架构;

② 基于多实例架构提出了支持传统进程线程模型的一致性存储空间构造的全局存储管理方法;

③ 面向大规模 CCNUMA 高可扩展性需求,提出了弱共享进程/线程模型;

④ 通过在真实 CCNUMA 系统上的运行和评测,表明 MIOS 具有良好的系统可扩展性.

1　技术需求和设计原则

银河深度并行计算机系统采用全系统 CCNUMA 体系结构,最大可实现 4096 处理器规模的单映像系统.用户要求银河深度并行计算机可以有效支持基于消息通信的 MPI 和基于共享存储的 OpenMP 并行应用高效运行,并能够对 CAF 等 PGAS 语言应用提供良好的性能加速比.为了有效管理大规模 CCNUMA 系统中的计算资源,提供高效、高可靠的计算服务,在分析和评测影响操作系统可扩展性关键因素的基础上,MIOS 采用了如下设计原则.

1) 数据显式共享原则:传统操作系统采用了隐式声明共享的数据布局方式,即所有数据缺省是共享的,这就导致大量共享数据管理开销,MIOS 采用了除特殊声明外,把用户数据和任务都作为本地信息管理的设计原则;

2) 利用硬件特性实现性能优化:为了性能优化,MIOS 牺牲了可移植性,充分利用银河深度并行计算机系统的硬件机制,大大提高了系统在任务间通信、临界区管理等方面的性能;

3) 充分考虑系统规模动态可调:为了支持不同规模下 MPI,OpenMP 应用的高效运行,MIOS 采用了根据系统规模动态构筑单映像运行环境的设计原则,可以有效满足不同规模下的系统灵活配置需求.

2　多实例操作系统结构

MIOS 操作系统的逻辑功能结构图如图 2 所示:

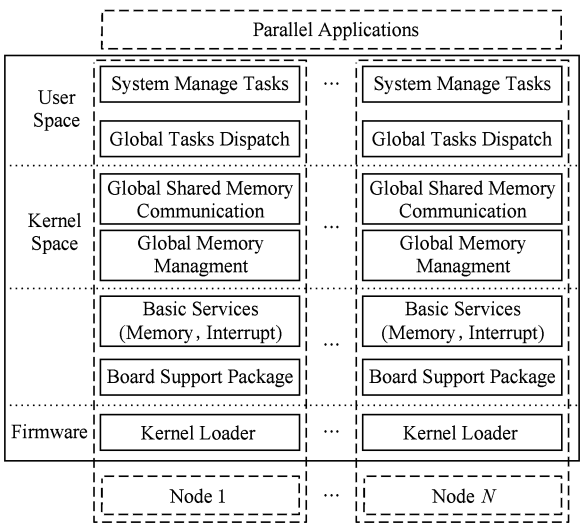


Fig. 2　Structure of MIOS operating system.

图 2　多核操作系统 MIOS 结构

1) 固件空间模块:由于 MIOS 采用了多处理器独立加载的启动方式,所以原来属于传统固件空间的内核加载功能被整合到操作系统,MIOS 的内核加载功能基于 UEFI 标准实现,提供了独立的 MIOS 动态浮动加载功能;

2) 内核空间模块:内核空间的 MIOS 模块包括基础服务模块和全局服务模块两类,其中基础服务模块提供了一个单操作系统实例内的设备管理和支持,以及中断处理、基础存储管理和任务调度等,全局服务模块在基础服务模块的基础上,提供了跨实例的全局一致的存储管理和共享通信服务等,通过

内核空间服务模块,MIOS 向用户提供了单一系统映像的运行环境;

3) 用户空间模块:用户空间模块包括全局任务分配模块和系统核心管理任务等,全局任务分配模块根据用户应用的任务数灵活分配计算资源,并加载任务,系统核心管理任务模块实施对本结点的管理和控制.

MIOS 的动态运行结构呈现多实例模式:CCNUMA 系统中每个结点上都运行代码相同但是独立运行的操作系统实例(如图 2 所示).各实例独立管理本结点的计算资源,也可根据需求灵活构成具有共享存储空间的可扩展单一系统映像运行环境,支持用户 MPI 和 OpenMP 任务的高效运行.

3 MIOS 实现

MIOS 基于 Linux 2.6.5 操作系统内核改造.为了有效提高 MIOS 在银河深度并行计算机系统上的可扩展性,提高用户应用的实际计算效率,我们不仅设计和实现了 MIOS 多实例架构,还分别针对影响操作系统可扩展性的锁、存储分配、任务加载/调度以及虚实空间映射管理等进行了针对性优化.

3.1 多内核浮动加载和启动

MIOS 采用多实例结构,启动时每个处理器独立加载一个 MIOS 的实例.当 MIOS 各实例运行到同步点后,协同构成全系统一致的资源管理视图.

银河深度并行计算机系统计算结点的存储空间采用全系统线性编址方式,即所有计算结点的存储空间统一顺序编址,这就导致每个结点加载 MIOS 实例时的加载空间互不相同.为了解决这个问题,MIOS 内核采用虚地址方式编址,通过浮动加载技术加载和运行各实例.加载流程如图 3 所示:

① Kernel loader collects physical memory space information of local node
② Calculate real loading address according to virtual address in MIOS image
③ Load image of MIOS to destination
④ Kernel loader configure the TLB mapping between virtual address and loaded address
⑤ Set the processor to virtual mode
⑥ Booting the MIOS kernel respectively
⑦ Start to access global resource until Synchronization of different MIOS instances

Fig. 3 Float loading scheme of MIOS.

图 3 MIOS 浮动加载方法

如果采用物理地址相关的传统操作系统加载方式,MIOS 就必须针对银河深度并行计算机的每个计算结点生成一个对应的内核映像.采用虚地址装配和浮动加载技术后,MIOS 屏蔽了物理地址的差异,每个结点都可以加载统一的 MIOS 内核,各计算结点实现了对相同内核虚地址空间到不同物理地址空间的映射和访问.

3.2 全局存储管理

全局存储管理是 MIOS 全局系统服务的核心.MIOS 的全局存储管理包括全局物理存储空间管理、全局虚空间管理和全局页故障处理 3 部分.

1) 全局物理存储空间管理

MIOS 采用了开放式物理存储管理框架:各个结点的内核实例加载时只构建本结点存储管理数据结构,如物理空间管理数据结构、页表管理数据结构和虚实映射管理数据结构等.当每个结点的 MIOS 实例运行到同步点后,通过全局系统服务模块将每个独立的管理数据结构动态链接成可管理全系统存储空间的全局管理数据结构(如图 4 所示).任何实例可以通过管理结构的逻辑环获取其他结点的物理存储空间信息,可以分配或者释放位于其他结点的物理内存,实现全局物理内存空间的管理.

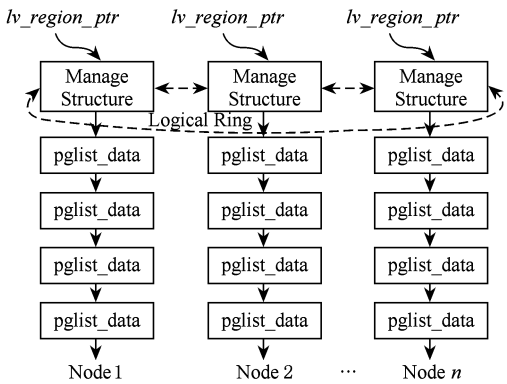


Fig. 4 Scalable manage ment structure of physical memory pages.

图 4 可扩展物理存储管理框架

2) 全局虚空间管理

MIOS 还向用户应用提供了跨内核实例的统一虚地址空间管理.为了提高访问虚地址管理空间的性能,虚地址管理数据结构(VMA)采用优先本地分配的方式均匀分布在各结点上,通过双向环构成统一的资源链.为了提高空间利用率,MIOS 的共享虚地址管理数据结构分配采用零售(slab)分配方式,通过自由和使用队列分别管理空闲和使用中的虚空间数据结构项.MIOS 提供了完整的共享虚地址管

理数据结构的初始化、分配和释放过程,并支持各实例对所有虚空间管理数据项的连接、插入和查找等过程.

3) 全局页故障处理

基于全局页故障处理功能,和其他操作系统的 NUMA 优化一样,MIOS 同样提供了多存储分配策略的支持.目前,MIOS 支持首次分配(first-touch)、指定结点以及跳跃式等其他复杂方式的数据分配,用户可以在分配数据时指定数据分配策略.

MIOS 中需要调用全局页故障处理的场景有 3 个:应用执行时的共享静态数据段分配、调用全局 malloc 函数时需要立即分配物理空间以及对先前全局 malloc 函数所分配空间进行访问时所引发的实际物理空间分配.由于高性能计算机一般不带本地磁盘,所以 MIOS 忽略了页交换的处理情况.MIOS 的全局页故障处理主要流程如图 5 所示:

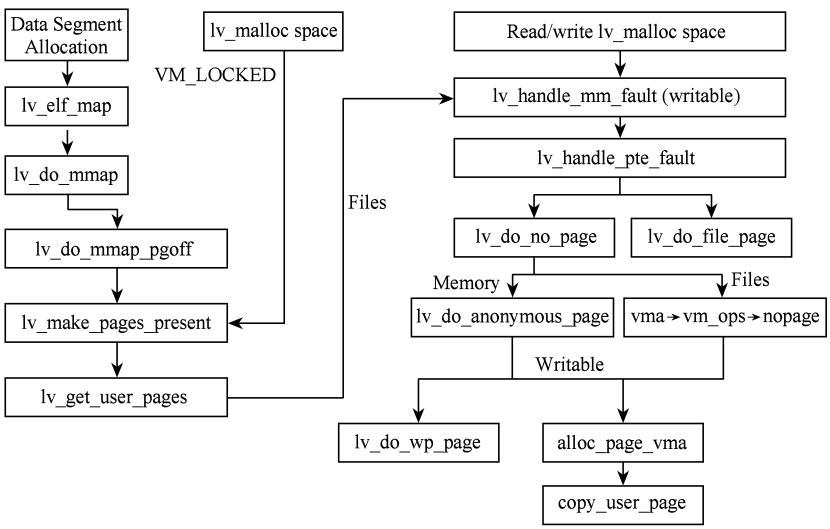


Fig. 5　Page fault processing for global memory.

图 5　全局页故障处理流程

对于指定结点分配策略,在用户请求分配虚空间时,MIOS 就立即在用户指定结点上分配真实物理内存;而对于 First-Touch 的存储分配策略,实际物理页面是在页面被访问时(如图 5 中右侧的处理流程),由 MIOS 调用全局页故障处理功能临时分配.First-Touch 策略保证了真实物理空间在第 1 次被访问时才真正分配,确保了用户数据的分布局部性.

3.3　弱共享线程模型和层次式任务调度

针对高性能计算应用的特点,并基于显式数据共享的设计原则,MIOS 采用了弱共享进程/线程并发模型,并设计了层次式任务调度机制.

传统线程模型定义,同属于一个进程的多个线程间可以共享执行代码空间以及私有数据外的其他数据空间,包括信号处理空间等.传统线程模型间的大量共享数据将导致并行应用在大规模 CCNUMA 系统上的可扩展性差.为此,基于显式数据共享的设计原则,MIOS 定义了新的弱共享线程模型:同属于一个 OMP 作业的各个线程间具有独立的执行代码、栈区和堆区,只有静态变量和显式定义的共享数据区是线程间共享的.采用弱共享线程模型可以有效制约 OMP 任务间的瓶颈数据共享访问,提高任务的可扩展性.

为了区分定义共享数据和私有数据,MIOS 重新定义了线程的装配地址区间.MIOS 的编译装配器将 Linux 操作系统中未使用的 4 区空间作为线程共享数据空间,将 3 区空间定义为线程私有数据空间.具体如图 6 所示.MIOS 的弱共享线程模型支持动态链接库.在应用运行加载动态链接库时,MIOS 将判断并将库文件放置到私有空间中.

针对弱共享进程/线程模型,MIOS 设计了层次式任务调度器.层次式任务调度器由各内核实例中的传统多域任务调度器和用户层的任务分配器构成.内核多域调度器支持在一个 MIOS 实例内调度任务;用户层的任务分配器提供了跨实例的任务分配.用户应用加载时,首先由任务分配器确定各任务的运行实例,然后由内核调度器调度执行.出于性能考虑,MIOS 不支持并行应用的进程/线程在各个实例间的动态迁移,从而回避了因迁移引发远程访问开销.

Assemble Space of Application on Linux		Assemble Space of Application on MIOS	
Comments	Region Number	Comments	Region Number
Kernel Region	7,6,5	Kernel Region	7,6,5
Unused Region	4	Shared Stack Region	STACK_TOP ▼
		Shared Brk Region	4 ▲
Stack Region Brk Region Data Region	STACK_TOP ▼ 3+data_end ▲ 3 ▲	Stack Region Brk Region Data Region	STACK_TOP ▼ 3+data_end ▲ 3 ▲
GOT,PLT	3	GOT,PLT	3
Text Region	2	Text Region	2
Mmap Region	1	Mmap Region	1

Fig. 6 Space layout of weak-shared thread executing model.

图 6 MIOS 弱共享线程地址空间布局

对于 MPI 并行应用,由于 MPI 任务各进程间具有独立的地址空间,并通过消息通信机制实现信息交换.因此,和传统集群系统类似,MPI 应用的各个任务是通过 MIOS 的用户层任务分配器部署到不同实例上加载执行的.对于 OMP 应用,MIOS 任务分配器首先确定要加载的实例,然后每个实例独立加载 OMP 线程.OMP 应用的各个线程可以访问由 MIOS 全局物理存储空间管理和虚空间管理所构建的共享数据存储区.在共享数据存储区中的数据支持所有线程间的共享访问.

采用弱共享线程模型和层次式调度技术,MIOS 提供了显式共享语义支持.虽然 MIOS 不支持线程在各个 MIOS 实例间的动态迁移,灵活性不足.但是反而因为在大规模 CCNUMA 系统中减小了大量远程数据访问,反而大大提高了并行应用的可扩展性.

3.4 自适应的任务间通信

MIOS 同时提供了面向短消息的基于共享区的低延迟通信机制和面向大消息的用户级高带宽通信机制.用户可以根据数据通信的大小灵活确定通信方式.

MIOS 的低延迟共享通信机制提供了跨内核实例的符合 POSIX 标准的 shared memory 共享语义通信.用户应用可以像使用 shared memory 共享区一样实现低延迟的小尺寸消息通信.

MIOS 的跨内核共享缓冲区模块由元数据管理和远程映射操作两部分构成,如图 7 所示.为了管理共享区缓冲区,MIOS 采用了元数据复制方法:在每个结点的固定地址开辟元数据存储区.当用户申请共享存储段时,元数据管理器将返回唯一的共享存储段标识.元数据管理功能同时将新创建的共享区元数据信息复制到其他结点的空闲元数据槽中.为了满足不同数据量通信的需求,MIOS 的跨内核实例 shared memory 可以支持最小 16 B、最大 16 MB 的共享存储段.跨内核实例的共享区模块提供了符合 POSIX 标准的共享区创建、删除以及远程映射、去映射等操作.

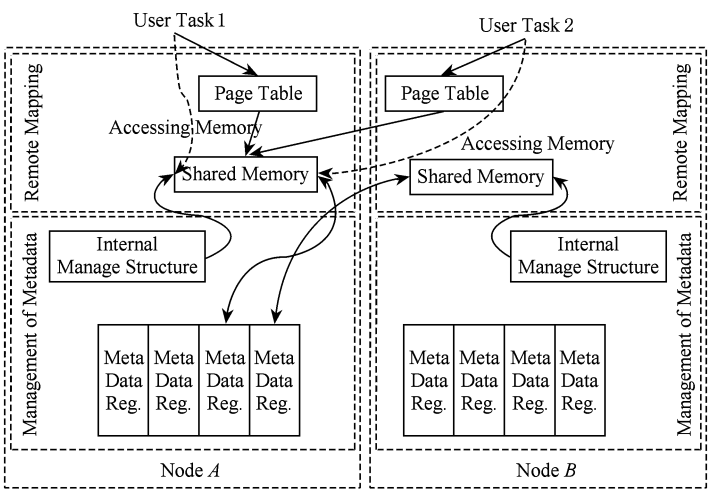


Fig. 7 Structure of shared memory crosssing OS.

图 7 跨 OS 的 shared memory 模块结构和实现原理

此外,MIOS 还提供了显式的高带宽用户层通信接口 GLEX.基于硬件的结点间互连控制芯片 NC,GLEX 用户层通信库提供了基于描述符编程界面的通信方式,支持结点间 RDMA 读写.GLEX 由

内核中的 NC 设备驱动程序、用户层的用户接口库和配置管理工具组成,支持基于端点的多进程并发、受保护的用户级通信操作,可以实现进程间数据块的零拷贝流水化传输.

3.5　寄存器锁优化

并行操作系统中的同步/互斥性能是影响操作系统性能和可扩展性的重要因素之一.传统操作系统基于内存变量,利用比较交换指令和取加指令实现旋转锁和读写锁等.在大规模 CCNUMA 系统中,当运行在不同结点的 N 个进程通过内存变量锁同步时,锁的竞争过程将产生 $O(N^2)$ 个 Cache 失效报文.也就是说,在 CCNUMA 系统中,操作内存变量锁会因为对特定内存的频繁读写而导致 Cache 抖动,其同步/互斥开销会随系统规模呈平方关系快速增长.MIOS 在传统内存变量锁的基础上,设计了基于硬件同步寄存器的全局寄存器锁模块,提供包括旋转锁和读写锁等功能.寄存器旋转锁算法如图 8 所示:

```
① Loop:
②   Val=Read(Base address of lock;addr_reg_base)
③   If(val==0) {
④       Val=Read(Base address of lock;addr_reg_base+10);
⑤       If(val==0) {
⑥         Required the spinlock
⑦       } else {
⑧         Goto Loop
⑨       }
⑩   } else {
⑪       Goto Loop
⑫ }
```

Fig. 8　Algorithm of spinlock based on synchronize register.

图 8　基于同步寄存器的旋转锁算法

同时,为了提高大规模并行应用的同步和规约操作性能,MIOS 通过系统映射技术将硬件同步寄存器映射到用户空间,为用户应用的高性能同步/互斥提供了实现基础.

3.6　TLB 虚实映射一致性维护

处理器的 TLB 寄存器缓存了用户应用的虚实空间映射关系.TLB 项的内容是运行进程上下文相关的.MIOS 采用多实例架构和层次式调度方式,各个内核独立调度任务.所以当修改一个处理器的 TLB 项时,其他处理器的 MIOS 很可能已调度了其他用户任务运行,TLB 的上下文发生变化.如果简单修改这些变化的 TLB 项就会导致各实例间的虚实映射关系混乱.为此,MIOS 设计了基于软切换的全局 TLB 一致性维护技术.

基于软切换的全局 TLB 一致性维护基于机间中断(IPI)和主动上下文切换技术实现.具体流程如图 9 所示:①当一个结点上的 MIOS 实例修改 TLB 映射关系时;②将首先修改本地的 TLB 内容;③然后通过机间中断(IPI)机制通知远程处理器进入 TLB 修改工作;④自己进入等待同步状态;⑤由于接收 IPI 时处理器可能发生了任务切换,因此首先要求切换到需一致性处理的任务环境,为了提高性能,MIOS 的主动上下文切换技术只是在 IPI 处理流程中切换了任务 TLB 寄存器中的任务环境,而没有通过操作系统调度器实现全任务切换,大大提高了处理性能;⑥各个处理器独立修改自己的 TLB 映射;⑦并通知发起者,以上的步骤⑤～⑦都是在 MIOS 的 IPI 处理中断时完成的;⑧当发起者发现所有结点完成 TLB 映射修改流程后退出本次 TLB 一致性维护流程.

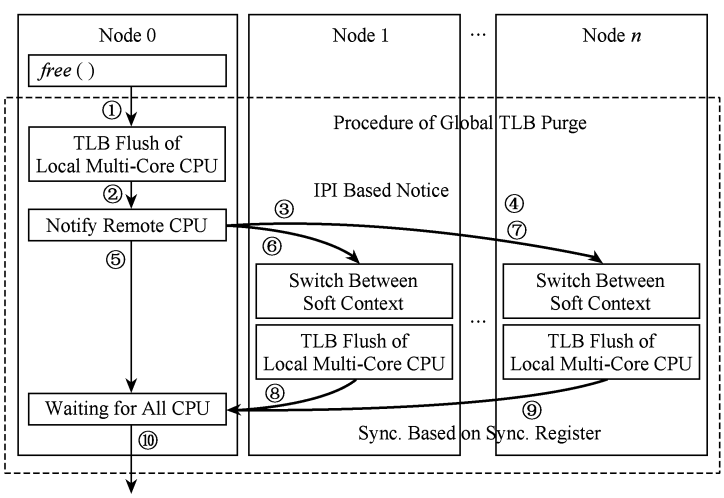


Fig. 9　Procedure of coherence maintaining of global TLB.

图 9　全局 TLB 一致性维护流程

4 评 测

本节测试了 MIOS 的可扩展性以及系统性能开销等. 测试包集合包括微 Benchmark, NASA 并行测试包和实际用户并行应用等. MIOS 的测试环境是银河深度并行计算机系统.

银河深度并行计算机系统采用 CCNUMA 体系结构, 系统由计算结点、服务结点和 IO 结点 3 部分组成. 银河深度并行计算机系统的计算结点由 2048 个处理器器构成. 每个计算结点含 8 个处理器、32GB 存储器和一个互连控制器 NC. 4 个互连控制器 NC 间通过一个互连路由器 NR 实现一级互联. 互连控制器 NC 提供 Cache 一致性协议, 全系统各个计算结点通过多级 NC 和 NR 互联构成紧耦合的具有统一线性地址空间的大规模 CCNUMA 系统.

4.1 操作系统微 Benchmark 测试

MIOS 采用多实例结构, 每个内核实例的运行环境都集中在一个结点上. 我们采用微 Benchmark 对比测试了 MIOS 和 Linux 操作系统的调用开销和 IO 开销等. 图 10 的测试结果表明 MIOS 每个实例的性能和 Linux 基本相当, 差距几乎可以忽略不计:

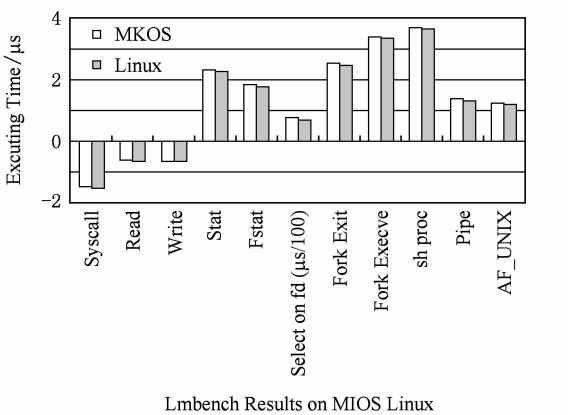


Fig. 10 Micor benchmkar testing on MIOS and Linux.

图 10 MIOS 和 Linux 的微 Benchmark 性能对比

4.2 TLB 一致性维护性能开销

由于 MIOS 各实例间采用了基于机间中断和快速上下文切换的全局 TLB 一致性维护. 由于银河深度并行计算机的机间中断 IPI 只能采用点对点的发送方式, 所以 TLB 的一致性维护开销随系统规模的增大而增长. 但是由于 TLB 维护流程中采用了流水方式对每个处理器的 TLB 项修改, 并采用了只恢复部分 TLB 上下文的快速切换技术, IPI 中断处理开销很小, 只有 20 μs 左右. 图 11 测试了不同系统规模下的 TLB 一致性维护开销. 测试表明 TLB 的维护开销主要随 CCNUMA 系统规模呈线性增长.

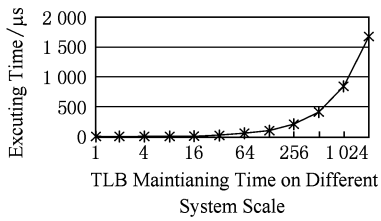


Fig. 11 Overhead of TLB maintianing.

图 11 TLB 维护开销

TLB 一致性维护的时机主要发生在页面去映射和释放页面时. 根据实际应用运行情况的统计, 在操作系统内核编译的 560 s 时间内, 平均每秒发生 TLB 维护请求 100 多次, 实测 TLB 总维护开销占运行总开销的 0.21%左右. 为了进一步提高性能, 可以考虑提供硬件实现的快速 TLB 一致性维护支持.

4.3 多存储分配策略性能评测

由于银河深度并行计算机系统规模庞大, 导致本地和远程访存差异大, 通过支持多种数据分配和放置方式可以有效提高应用的数据局部性. MIOS 支持 First-Touch、指定结点和跳跃分配等多种分配方式. 表 1 显示了应用在 16 个任务规模下使用不同数据放置方式后的性能差异:

Table 1 Performance of Different Allocating Scheme

表 1 不同存储分配策略的性能优化 s

Applications	First-Touch	Designate Allocate	Nested Allocate
ft. C	75.86	218.94	
Preferred		1.961 742	13.360 125
Interleave		10.026 7	3.526

从表 1 可以发现, ft. C 应用使用 First-Touch 方式的性能可以比固定某一结点分配数据时性能高 3 倍. 在 Preferred 应用中, 指定结点方式可以比随意的跳跃分配性能提高 5 倍以上, 而 Interleave 应用的测试又有相反的结论. 测试表明, 针对应用特点使用适合的数据放置策略可以有效提升应用性能.

4.4 寄存器锁可扩展性评测

图 12 示出采用寄存器锁和传统内存锁后的系统加锁性能对比. 当对一个锁变量的竞争任务数小于 8～16 时, 内存锁技术由于 Cache 读写速度高于非 Cache 的寄存器访问, 所以加锁延迟性能较好. 但是当竞争者超过 16 时, 由于 Cache 行抖动, 内存锁延迟迅速增长. 而寄存器锁采用了排队访问方式, 消除了 Cache 抖动带来的不利影响, 所以即使系统规模增加, 寄存器锁延迟可以基本保持稳定. 当竞争数

达到 2 048 时,寄存器锁的性能可以较内存锁提高约 40 倍.

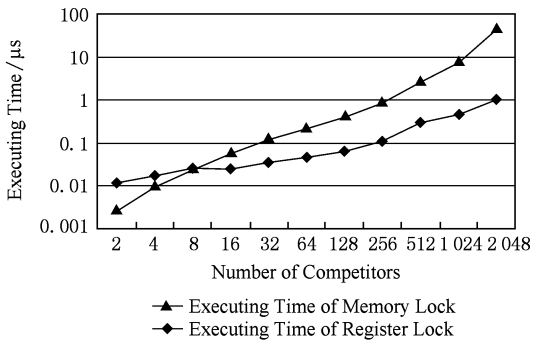


Fig. 12　Performance of register lock and memory lock.

图 12　寄存器锁和内存锁的性能对比

寄存器主要应用在可能发生大量竞争者的环境中,如物理内存页锁、虚空间结构锁等.而内存锁则主要应用在竞争者少的情况下,如局部调度队列锁等.

4.5　进程间通信性能评测

图 13 示出基于共享缓冲区的低延迟通信和标准通信方式 GLEX 间的传输延迟对比.由于跨内核的基于共享缓冲区的通信方式无中断处理流程,所以当通信数据尺寸小于 256 B 时,该方式的通信延迟只有标准用户层通信方法 GLEX 的一半左右.但是当通信数据量增大时,数据需要通过多个 Cache 行的传输,共享缓冲区通信方式的延迟将快速上升.而 GLEX 采用大数据块的传输方式,数据块的尺寸达 8 KB.所以,测试表明当通信数据大于 256 B 时,用户层通信库 GLEX 的通信延迟和带宽都将优于共享通信方式.用户可以根据自己的通信需求灵活选择适合的通信方式.

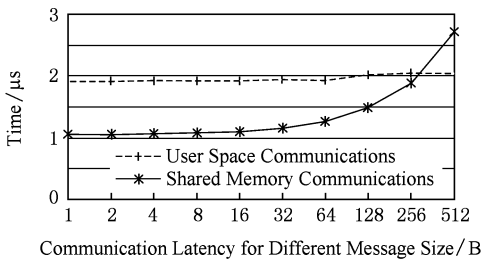


Fig. 13　Performance of shared communitions and user space communications.

图 13　共享通信和用户层通信延迟对比

4.6　系统可扩展性评测

系统可扩展性评测显示了不同规模不同类型应用在 MIOS 上的性能加速比.

针对 OMP 应用,虽然 OMP 采用多线程并发模型,但是由于 MIOS 采用了弱共享线程模型和层次式调度,应用的正文段、私有数据段局部化放置,只有显式共享数据才能全局共享、动态分配.再加上利用适合存储分配策略等优化,使 OMP 应用在大规模 CCNUMA 系统上仍有良好的加速比.

图 14 示出不同规模的二维欧拉程序 MFIC 应用的加速比.规模 3 的 MFIC 任务数从 16 开始扩展到 256,处理器使用数从 8 个扩展到 128 个.规模 5 的 MFIC 任务数从 256 开始扩展到 4 096,处理器使用数从 256 个扩展到 2 048 个.在规模 3 下,MFIC 呈现出超线性加速比,当规模扩展到 2 048 处理器时,MIOS 对 MFIC 应用仍提供了较好的可扩展性支持.

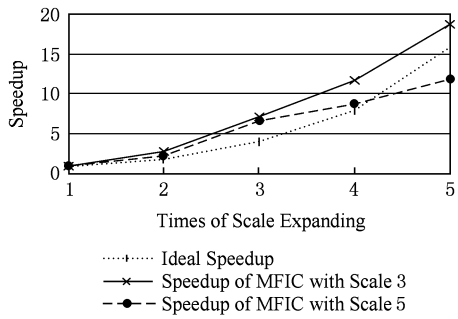


Fig. 14　Speedup of different scale of MFIC application.

图 14　MFIC 的不同规模性能加速比

图 15 示出不同 OMP 应用在 MIOS 上的性能加速比.横轴采用 log 方式表示.由于不同 OMP 应用的规模不同,所以我们只能测试出在可行规模下的 MIOS 可扩展能力.从图 15 可以发现,OMP 应用在 MIOS 可以呈现良好的性能可扩展性.

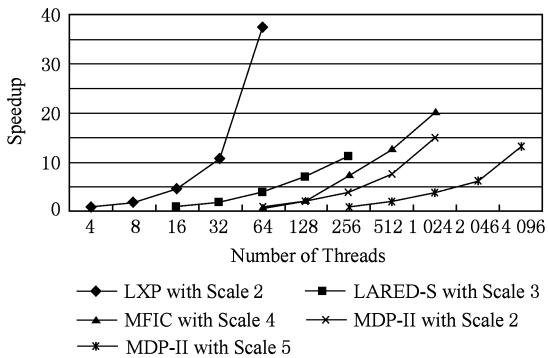


Fig. 15　Speedup of differnet OMP applications.

图 15　不同 OMP 应用的性能加速比

表 2 示出在 MIOS 上和标准 Linux 操作系统上运行不同 MPI 应用 10 次后的平均性能对比测试.从表 2 可以发现,由于 MPI 基于进程级并行,MIOS 把 MPI 的每个进程固定到不同内核上运行.并且每

个进程都只使用本结点的局部存储空间,所以在MIOS上的MPI应用性能和采用独立Linux操作系统构成的集群环境具备类似的性能,平均差异在3%左右.

Table 2 Performance of MPI Applications on MIOS and Linux

表2 MIOS和Linux上MPI应用性能差异

Applications	Executing Time on MIOS/s	Executing Time on Linux/s	Difference /%
LXP	138	130	−5.79
MFIC	4 832	4 852	4.13
MC	2 231	2 290	2.64
LARED-P	14 332	14 132	−1.39
LARED-S	6 213	6 315	1.64

图16示出了MIOS上运行CAF类应用的性能.由于MIOS采用了显式共享的设计理念,对基于PGAS语言的应用具有固有的良好支持力.CAF语言是PGAS语言中的一种.通过评测CAF语言应用,我们可以发现MIOS同样可以呈现良好的性能可扩展性.

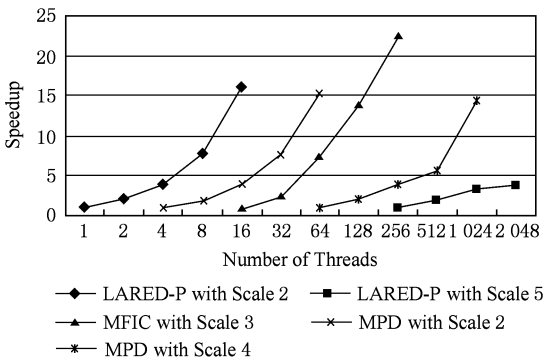


Fig. 16 Speedup of CAF applications.

图16 CAF语言应用的性能加速比

5 结论和下一步工作

MIOS针对大规模CCNUMA系统的特点和可扩展性需求设计,采用了多实例架构和新型弱共享线程模型,设计了高效的全局存储管理,并提供了低延迟高带宽通信机制和寄存器锁优化等.测试表明MIOS在大规模CCNUMA系统上具有较好的可扩展性.MIOS于2007年研制成功,并通过鉴定.

目前随着硬件技术的飞速发展,在一个芯片上集成数十乃至数百上千的处理器核心已成为未来多核处理器系统发展的主流方向.采用传统单内核操作系统将无法满足大规模多核处理器系统的可扩展性需求.MIOS的多实例架构对于大规模多核处理器系统也有良好的支持.

下一步,MIOS将结合面向多核系统的语言技术发展,对未来的新型语言和应用提供更高效的支持.

参 考 文 献

[1] Mall M. AIX suport for memory affinity [R]. New York: IBM Corporation, 2002

[2] Bligh M J, Dobson M, Hart D, et al. Linux on NUMA system [EB/OL]. [2010-03-12]. http://www.kernel.org/pub/linux/kernel/people/mbligh/presentations/OLS2004-numa_paper.pdf

[3] Bryant R, Hawkes J. Linux scalability for large NUMA systems [EB/OL]. [2010-03-12]. http://oss.sgi.com/projects/numa/Linux_Scalability_for_Large_NUMA_OLS2003.pdf

[4] Frank H, Nagar S, Kravetz. PMQS: Scalable Linux scheduling for high end servers [EB/OL]. [2010-03-12]. http://www.usenix.org/publications/library/proceedings/als01/franke.html

[5] Kravetz M, Franke H. Enhancing Linux scheduler scalability [EB/OL]. [2010-03-12]. http://lwn.net/2001/features/OLS/pdf/pdf/elss.pdf

[6] Lameter C. Local and remote memory: Memory in a Linux/NUMAsystem [EB/OL]. [2010-03-12]. http://www.kernel.org/pub/linux/kernel/people/christoph/pmig/numamemory.pdf

[7] Kleen. A Numa API for Linux [EB/OL]. [2010-03-12]. http://www.novell.com/collateral/4621437/4621437.pdf

[8] Intel QuickPath Technology [EB/OL]. [2010-03-12]. http://www.intel.com/technology/quickpath

[9] AMD HyperTransport Technology [EB/OL]. [2010-03-12]. hypertransport-technology/Pages/hypertransport-technology.aspx

[10] Wells C K. Computation spreading: Employing hardware migration to specialize CMP core on-the-fly [C] //Proc of the 12th Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2006: 283-292

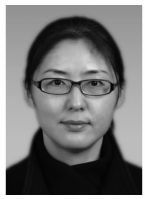
[11] Nightingale E B. Helios: Heterogeneous multiprocessing with satellite kernel [C] //Proc of the ACM SIGOPS 22nd Symp on Operating Systems Principles. New York: ACM, 2009: 221-234

[12] Baumann A. The Multikernel: A new OS architecture for scale multicore systems [C] //Proc of the ACM SIGOPS 22nd Symp on Operating Systems Principles. New York: ACM, 2009: 29-44

[13] Wentzlaff D. Factored operating system (fos): The case for a scalable operating system for multicores [J]. ACM SIGOPS Operating System Review, 2009, 43(2): 76-85

Lu Kai, born in 1973. Received his PhD degree in computer science from the National University of Defense Technology in 1999. Currently, professor and PhD supervisor of NUDT. His current research interests include embedded and large scale parallel system software.

Chi Wanqing, born in 1973. Received his master degree in computer science from the National University of Defense Technology in 1998. Currently, he is associate professor and master supervisor in NUDT. His current research interests include embedded and large scale parallel system software.



Gao Yinghui, born in 1975. Received her PhD degree in signal processing from the National University of Defense Technology in 2006. Currently, Lecturer in NUDT. Her current research interests include embedded and large scale parallel system software.

Feng Hua, born in 1976. Received his master degree in computer science from the National University of Defense Technology in 2003. Currently, he is assistant professor in NUDT. His current research interests include embedded and large scale parallel system software.