A Synopsis of the Process Scheduling Challenges in the Era of Multi-Core Processors

多核处理器时代进程调度的挑战概述

> Abstract – A number of process scheduling techniques have been proposed and thoroughly evaluated in the literature for multi-core processors. Each of these techniques has its own unique attractive features and challenges. This synopsis paper discusses how innovations in multi-core processors bring new challenges to the process scheduler and how it optimizes the process scheduling in these environments. An example of Linux is also presented which describes how Linux process scheduler handles these challenges and design innovations. Power management issues in multi-core processors and Intel Dynamic Acceleration technology, discussed in original article, are beyond the scope of this synopsis.

摘要 - 已经提出了许多过程调度技术，并在文献中对多核处理器进行了全面评估。这些技术中的每一种都有其独特的吸引人的特征和挑战。本概要文章讨论了多核处理器的创新如何为流程调度程序带来新的挑战，以及如何优化这些环境中的流程调度。还介绍了Linux的一个示例，其中描述了Linux进程调度程序如何处理这些挑战和设计创新。原始文章中讨论的多核处理器和英特尔动态加速技术中的电源管理问题超出了本概要的范围。

> Keywords: multi-core processors, process scheduling

关键词：多核处理器，进程调度

I. INTRODUCTION

> The process scheduler is an essential piece of operating system software that manages the CPU resource allocation to tasks in a way which maximizes system throughput, minimizes the response time and ensures fairness among the running tasks in the system. In multi-core processor packages, each processor package contains two or more execution cores, with each core having its own resources (registers, execution units, some or all levels of caches, etc.). Design innovations of multi-core processor architectures mainly span the area of shared resources (caches, power management, etc.) between cores, core topologies (number of cores in a package, relationship between them, etc.), and platform topology (relation between cores in different packages, etc.). These innovations bring new opportunities and challenges to the process scheduler.

进程调度程序是操作系统软件的重要组成部分，它以最大化系统吞吐量，最小化响应时间和确保系统中正在运行的任务之间的公平性的方式管理任务的CPU资源分配。 在多核处理器包中，每个处理器包包含两个或更多个执行核，每个核具有自己的资源（寄存器，执行单元，一些或所有级别的高速缓存等）。多核处理器体系结构的设计创新主要跨越核心之间的共享资源（高速缓存，电源管理等）领域，核心拓扑（封装中的核心数量，它们之间的关系等）和平台拓扑（关系） 在不同包中的核心之间等）。 这些创新为流程调度程序带来了新的机遇和挑战。

II. OBJECTIVES

> Suresh et al. conducted this study to look at how different multi-core topologies bring new optimization opportunities to the process scheduler. They examined these scheduling mechanisms to analyze how these are implemented in the Linux process scheduler.

Suresh等人进行了这项研究，以研究不同的多核拓扑如何为进程调度程序带来新的优化机会。他们检查了这些调度机制，以分析如何在Linux进程调度程序中实现这些机制。

III. METHODOLOGY

> At first, the authors have discussed how earlier challenges in the Symmetric Multiprocessing (SMP), Non-Uniform Memory Access (NUMA) and Symmetric Multithreading (SMT) were addressed. Then they expressed multi-core topologies with respect to core, cache, and platform topologies. Finally, they moved on to a symmetric multi-core aware process scheduler and examined different scheduling mechanisms in this area under an experimental setup comprising of a 3GHz dual-package SMP platform with each package having two cores sharing a 4MB last-level (L2) cache, without the support of Intel® Dynamic Acceleration Technology [1]. For their analysis, the authors tested scheduling mechanisms using different workloads such as SPECjbb2000, SPECjbb2005, SPECfp_rate of CPU2000 and an In-Memory Database Search (IMDS). Suresh et al. examined the need for a multi-core-aware process scheduler. Then different scheduling mechanisms for multi-core platforms under different load scenarios and the associated tradeoffs were discussed. Authors also discussed how some of these scheduling mechanisms are currently implemented in the Linux operating system. Finally, they closed their study with a discussion of future research opportunities in this field.

首先，作者讨论了如何解决对称多处理（SMP），非统一内存访问（NUMA）和对称多线程（SMT）中的早期挑战。 然后，他们就核心，缓存和平台拓扑表达了多核拓扑。 最后，他们转向了一个对称的多核感知进程调度程序，并在一个包含3GHz双包SMP平台的实验装置中检查了该区域中的不同调度机制，每个包具有两个内核，共享一个4MB的最后一级（L2） 缓存，没有英特尔®动态加速技术的支持[1]。 对于他们的分析，作者使用不同的工作负载测试了调度机制，例如SPECjbb2000，SPECjbb2005，CPU2000的SPECfp_rate和内存数据库搜索（IMDS）。 Suresh等人。 检查了对多核感知进程调度程序的需求。 然后讨论了不同负载情况下多核平台的不同调度机制以及相关的权衡。 作者还讨论了当前如何在Linux操作系统中实现这些调度机制。 最后，他们结束了他们的研究，讨论了该领域未来的研究机会。

> A. Symmetric Multiprocessing (SMP)
>
> In the multi-core processor packages, the execution cores in a given processor package are symmetric. In SMP, each processor is self scheduling. All processors may be in a common ready queue, or each process may have its own private queue for ready processes. If multiple processors are trying to access and update the common data structure then the process scheduler must be programmed carefully [2]. B. The Process Scheduler
>
> Process priority determines the allotted time (time-slice) of a process on a CPU and when to run it on a CPU. In SMP domain, the process scheduler is also responsible for distributing the process load to different CPUs in the system. In NUMA platforms, memory access time is not uniform across all the CPUs in the system and depends on the memory location relative to a processor. System software tries to minimize the access times by allocating the process memory on the node that is closest to the CPU on which the process is running. In SMT, such as Intel® Hyper Threading Technology, most of the core execution resources are shared by more than one logical processor. In this case, the process scheduler needs to be aware of SMT topology and avoid situations where more than one thread sibling on one core is busy, while all the thread siblings on another core are completely idle.
>
> C. Multi-core Topologies

> In multi-core topologies, cores in a physical processor package share some of the resources e.g. Intel® Core-2 Duo processor has two processor cores sharing the same single L2 cache, as shown in Figure 1. Similarly, the Intel® Quad core processors have four cores in a single package with two lastlevel (L2) caches, and each of these L2 caches is shared by two cores.
>
> D. Multi-Core Scheduling
>
> Shared resource topologies in multi-core platforms pose interesting challenges and opportunities to the process scheduler. A fair amount of CPU time allocated to each task by the process scheduler will not essentially translate into efficient and fair usage of the shared resources. The main challenge before the process scheduler is to identify and predict the resource needs of each task and schedule them in a fashion that will minimize shared resource contention, maximize shared resource utilization, and exploit the benefits of shared resources between cores.

A.对称多处理（SMP） 在多核处理器封装中，给定处理器封装中的执行核是对称的。在SMP中，每个处理器都是自调度的。 所有处理器可以处于公共就绪队列中，或者每个进程可以具有其自己的专用队列以用于就绪进程。 如果多个处理器试图访问和更新公共数据结构，则必须仔细编程进程调度程序[2]。 B.进程调度程序 进程优先级确定CPU上进程的分配时间（时间片）以及何时在CPU上运行它。 在SMP域中，进程调度程序还负责将进程负载分配给系统中的不同CPU。 在NUMA平台中，系统中所有CPU的内存访问时间并不一致，并且取决于相对于处理器的内存位置。 系统软件尝试通过在最靠近运行进程的CPU的节点上分配进程内存来最小化访问时间。 在SMT中，例如英特尔®超线程技术，大多数核心执行资源由多个逻辑处理器共享。 在这种情况下，进程调度程序需要了解SMT拓扑，并避免一个核心上有多个线程兄弟忙，而另一个核心上的所有线程兄弟都完全空闲的情况。 C.多核拓扑 在多核拓扑中，物理处理器包中的核共享一些资源，例如： 英特尔®酷睿2双核处理器有两个处理器内核共享相同的单个二级高速缓存，如图1所示。同样，英特尔®四核处理器在一个包中有四个内核，带有两个最后级（L2）高速缓存，每个 这两个L2缓存由两个内核共享。 D.多核调度 多核平台中的共享资源拓扑为流程调度程序带来了有趣的挑战和机遇。 由进程调度程序分配给每个任务的相当大的CPU时间实质上不会转化为共享资源的有效和公平使用。 进程调度程序之前的主要挑战是识别和预测每个任务的资源需求，并以最小化共享资源争用，最大化共享资源利用率以及利用核心之间共享资源的优势的方式对其进行调度。

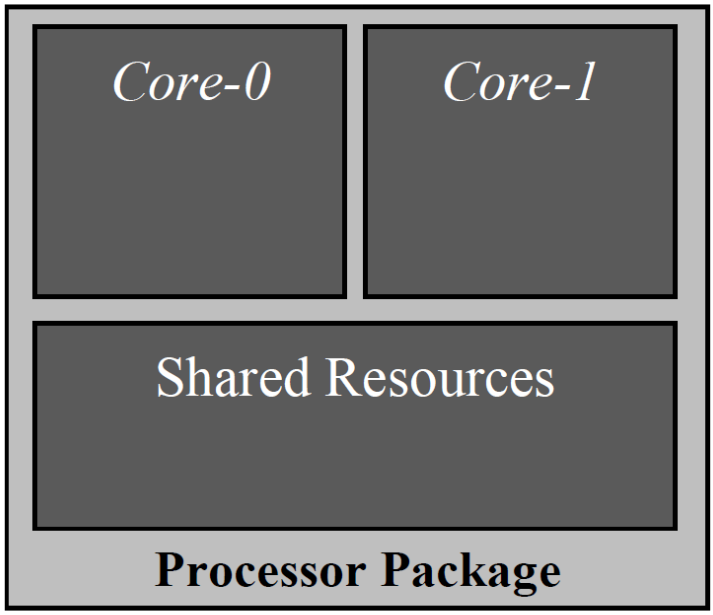

Fig. 1. Dual-core package with shared resources

IV. OUTCOMES

> The authors conducted experiments by running different workloads in multiple scenarios. These are narrated in the following lines.
>
> A. Scheduling on Cores Sharing Resources vs. Not Sharing.
>
> In this scenario, all the workloads were run in a 2-task configuration. The SPECjbb was run in two warehouse arrangements (where each warehouse represents two userlevel threads), and SPECfp_rate was run in the base configuration (base2000) with two users (where each user represents an individual process). The IMDS workload used two threads (belonging to the same process) to process the database queries. The results shown that all the workloads benefited by distributing the load to two different processors. Table I shows performance difference between scheduling two tasks running on two cores using different last-level cache vs. scheduling on cores sharing same last-level cache. Higher percentage means that scheduling on cores with different caches is better. Results also shown that contention is present whether the running tasks belong to the same process (i.e. with some data sharing e.g. SPECjbb) or different processes (i.e. without data sharing e.g. SPECfp_rate of cpu2000).

TABLE I Performance Difference Between Scheduling Two Tasks Running on Two Cores Using Different Last-Level Cache and Same Last-Level Cache

Workloads	%Performance improvement
SPECjbb2005	13.22
SPECjbb2000	5.19
SPECfp_rate(base2000)	16
IMDS	1

> The IMDS workload performed the same whether the threads running on cores with sharing the same or different packages. The reason being that the IMDS workload does not reveal good locality of memory references and thus not gets affected by sharing or not sharing the last-level cache between two threads.

作者通过在多种情况下运行不同的工作负载来进行实验。这些在以下行中叙述。 A.核心共享资源与不共享的调度。 在此方案中，所有工作负载都以2任务配置运行。SPECjbb在两个仓库安排中运行（每个仓库代表两个用户级线程），SPECfp_rate在基本配置（base2000）中运行，有两个用户（每个用户代表一个单独的进程）。IMDS工作负载使用两个线程（属于同一进程）来处理数据库查询。 结果显示，通过将负载分配给两个不同的处理器，所有工作负载都受益。表I显示了使用不同的最后一级缓存调度在两个核上运行的两个任务与共享相同的最后一级缓存的核上的调度之间的性能差异。更高的百分比意味着在具有不同缓存的核心上进行调度更好。 结果还表明，无论运行任务是属于同一进程（即，某些数据共享，例如SPECjbb）还是不同进程（即没有数据共享，例如cpu2000的SPECfp_rate），都存在争用。

无论在核心上运行的线程是否共享相同或不同的包，IMDS工作负载都执行相同的操作。原因是IMDS工作负载没有显示内存引用的良好位置，因此不会受到共享或不共享两个线程之间的最后一级缓存的影响。

> B. Task Group Scheduling.
>
> The authors argued that if all the running tasks are resource intensive, the challenge before the process scheduler is to identify the tasks that share data and schedule them on the cores sharing the same last-level cache. This will help minimize the shared resource contentions and shared data duplication because the system software has some inherent knowledge about data sharing between tasks. Similarly, processes attached to the same

> shared memory segment will share the data in that segment. The authors assumed that the process scheduler could group the threads belonging to a process or processes attached to the same shared memory segment and co-schedule them in the cores sharing the package resources. The authors ran two instances of the SPECjbb workload (SPECjbb2005 and SPECjbb2000), with each instance having two warehouses (each warehouse represented by a thread) on the experimental platform. The results shown that grouping the threads, belonging to a process onto the same package, takes advantage of shared resources between cores and improves performance, as depicted in Table II.

TABLE II Performance Improvement when Threads belonging to a Process are grouped

Workloads	%Throughput improvement
SPECjbb2005	10
SPECjbb2000	7.5

B.任务组调度。 作者认为，如果所有正在运行的任务都是资源密集型的，那么进程调度程序之前的挑战就是识别共享数据的任务，并在共享相同的最后一级缓存的核心上安排它们。这将有助于最小化共享资源争用和共享数据重复，因为系统软件具有关于任务之间的数据共享的一些固有知识。同样，连接到同一共享内存段的进程将共享该段中的数据。 作者假设进程调度程序可以将属于一个或多个进程的线程分组到同一共享内存段，并在共享包资源的内核中共同调度它们。作者运行了两个SPECjbb工作负载实例（SPECjbb2005和SPECjbb2000），每个实例在实验平台上有两个仓库（每个仓库由一个线程代表）。结果显示，将属于某个进程的线程分组到同一个包中，可以利用内核之间的共享资源并提高性能，如表II所示。

V. MULTI-CORE SCHEDULING IN LINUX

> The authors stated that in Symmetric Multiprocessing (SMP) domain, the Linux kernel scheduler evenly distributes the running tasks among the available last-level caches. For verification, they considered a dual processor package SMP platform with Intel® Core 2 quad processors with four running tasks. The Linux process scheduler distributed these four tasks among the four L2 caches available in the system, as shown in Figure 2. This lead to maximized resource utilization and minimized resource contention and, hence, provided optimal performance when tested on previously described workloads.
>
> The study shown that the Linux kernel provides a tunable that can be set by an administrator. When this tunable is set, the process scheduler will first try to load all the logical threads and cores in the package before distributing the load to another package. This policy is referred to as power-saving policy and provides optimal power saving. For example, when the authors ran the same four-task scenario on a dual package SMP platform Intel Core 2 Quad processors with power-saving tunable set, all the four tasks were running on the four cores residing in a single package, as shown in Figure 3.

作者表示，在Symmetric Multiprocessing（SMP）域中，Linux内核调度程序在可用的最后一级缓存中均匀分配正在运行的任务。为了验证，他们考虑使用具有Intel®Core2四核处理器和四个运行任务的双处理器封装SMP平台。Linux进程调度程序将这四个任务分布在系统中可用的四个L2高速缓存中，如图2所示。这样可以最大限度地提高资源利用率并最大限度地减少资源争用，从而在以前描述的工作负载上进行测试时提供最佳性能。该研究表明，Linux内核提供了一个可由管理员设置的可调参数。 设置此可调参数后，进程调度程序将首先尝试加载包中的所有逻辑线程和核心，然后再将负载分发到另一个包。该策略称为节电策略，可提供最佳节能效果。例如，当作者在具有节能可调谐集的双封装SMP平台英特尔酷睿2四核处理器上运行相同的四任务场景时，所有四个任务都在驻留在单个封装中的四个核上运行，如图所示图3。

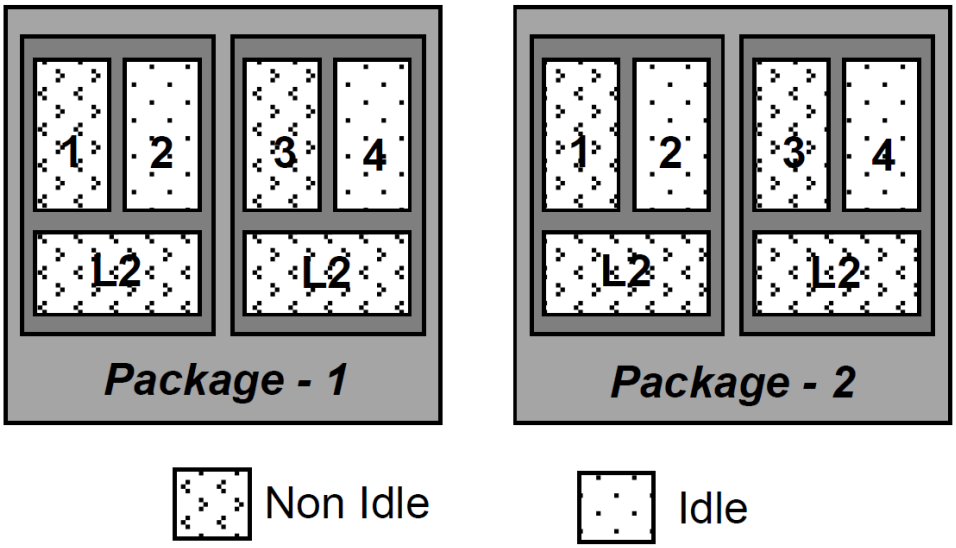

Fig. 2. Scheduling mechanism showing four running tasks scheduled on four L2s on a dual package with Intel Core 2 Quad processors

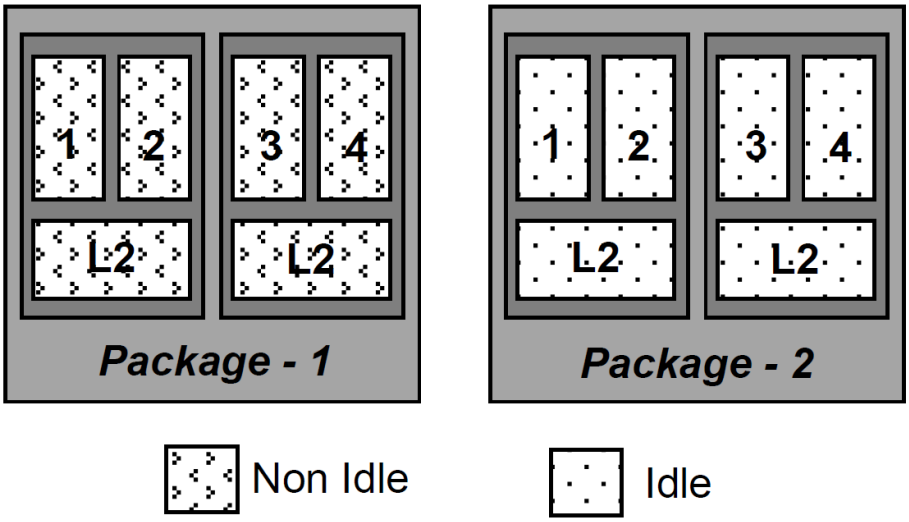

Fig. 3. Scheduling mechanism showing four running tasks scheduled on four cores in one single package on a dual package Intel Core 2 Quad processors with power-saving tunable set.

VI. FUTURE SCHEDULING CHALLENGES

> Suresh et al. suggested that for utilization of optimal performance, the process scheduler needs to schedule tasks in such a way that all the platform resources are used effectively. Moreover, this mechanism depends on the workload, processor, platform topology, and system load. They claimed that as more cores are incorporated in the processor package, the available shared resources will also increase accordingly. *The challenge for the process scheduler is to track the shared resource usages and the associated contentions.* In a situation where all the shared resources and processor packages are busy, the process scheduler will have to minimize the resource contention for optimal performance. The authors suggested that this can be achieved by grouping CPU-intensive and memory-intensive tasks onto the cores sharing the same last-level cache. They recommended using micro-architectural history also for predicting the behavior and characteristics of tasks and if such history is not

> available the system software can use some heuristics to estimate the resource requirements. One such heuristic may include number of physical pages accessed.

Suresh等人建议，为了利用最佳性能，进程调度程序需要以有效使用所有平台资源的方式调度任务。此外，此机制取决于工作负载，处理器，平台拓扑和系统负载。他们声称，随着处理器包中包含更多核心，可用的共享资源也将相应增加。 *进程调度程序的挑战是跟踪共享资源的使用情况和相关的争用。*在所有共享资源和处理器包都很忙的情况下，进程调度程序必须最小化资源争用以获得最佳性能。 作者建议，这可以通过将CPU密集型和内存密集型任务分组到共享相同的最后一级缓存的核心来实现。他们建议使用微架构历史来预测任务的行为和特征，如果这些历史不可用，系统软件可以使用一些启发式方法来估计资源需求。一种这样的启发式可以包括访问的物理页面的数量。

VII. SUMMARY

> In this study, the authors concluded that the process scheduler needs to be made aware of the multi-core topologies and task characteristics to obtain optimal performance. The authors analyzed multi-core scheduling mechanisms and challenges in an SMP environment. They proposed groupscheduling heuristics that can help to achieve the desired goals. They also provided a glimpse in how some of these multi-core scheduling mechanisms are implemented in the Linux operating system. Suresh et al. predicted that the future process scheduler will have consciousness of micro-architectural characteristics to attain optimal performance. They anticipated that future research direction in this domain will drive hardware and platform designs that will minimize the effects of shared resource contention and also assist the system software design in making and enforcing the right decisions.

在本研究中，作者得出结论，需要使流程调度程序了解多核拓扑和任务特征，以获得最佳性能。作者分析了SMP环境中的多核调度机制和挑战。他们提出了群体调查启发式方法，可以帮助实现预期目标。他们还提供了一些如何在Linux操作系统中实现这些多核调度机制的一瞥。 Suresh等人预测未来的流程调度程序将具有微架构特征的意识，以获得最佳性能。他们预计，该领域未来的研究方向将推动硬件和平台设计，以最大限度地减少共享资源争用的影响，并协助系统软件设计制定和执行正确的决策。