湖南大学

硕士学位论文

同构多核处理器的实时混合调度算法研究

姓名：王艳

申请学位级别：硕士

专业：计算机科学与技术

指导教师：李仁发

20120526

摘 要

近年来，多核处理器因其高性能、低功耗的特点在实时系统中得到了广泛应用。处理器架构的转变，给实时操作系统中任务调度的研究带来了新的挑战。

本文对多核混合调度算法进行了深入研究。纯局部调度不能充分利用核的剩余处理能力，同时全局调度因为频繁迁移造成开销过大，针对上述的问题，提出了一种改进的混合调度算法，并对其利用率进行了分析。另一方面，考虑到共享高速缓存对调度的影响，在软实时应用的背景下，提出了一种面向共享缓存改进的多核混合调度算法。本文的具体研究如下：

首先，本文在典型全局调度算法 EDF-US[1/2]的基础上提出了采用分簇策略的混合调度算法。该算法对同构多核处理器中的多个核平均分组，令同一个分组中的处理器核为一个处理器簇集。将待调度的任务按利用率从大到小排列，基于最差适应算法将任务分配到各个处理器簇集，并保证每个簇集内利用率大于 1/2 的任务不超过簇集内的核数。任务分配后在各个簇集内使用全局 EDF-US[1/2]调度算法，调度过程中，任务不允许在各个簇集间迁移。本文给出了采用分簇策略的混合调度算法的利用率下界，并分析比较了该混合调度算法与全局调度算法和局部调度算法的性能。

其次，为了减小调度过程中共享高速缓存的抖动次数，提出了一种改进的混合调度算法。通过建立有任务依赖的任务模型，采用启发式算法将存在依赖的任务分配到同一个共享二级缓存的处理器簇集；并在处理器簇集内鼓励这些任务共同调度。仿真研究表明改进算法相比于原算法可以使共享二级缓存的命中率得到提高，从而提高调度的效率。

关键词：同构多核处理器；实时调度；混合调度；算法利用率；共享高速缓存

Abstract

In recent years, Multicore architecture has been widely adopted in real-time system due to the higher processor performance and lower power consumption. Multicore architecture is different from the single-core architecture, so how to effectively exploit the parallelism in multicore platform brings new changes to real-time scheduling.

In this paper we made a deep research on hybrid approaches which combines elements of both partitioned and global scheduling. Fully partitioned approaches suffer from the drawback that the available processing capacity can become fragmented, such that although in total a large amount of capacity is unused. Meanwhile, the overheads incurred by global scheduling can potentially be very high. Neither partitioned nor global strategies could dominate over the other, so we presented an improved hybrid scheduling algorithm, and given a lower bound on the utilization bound of this algorithm. Then we analysed the hybrid scheduling through the utilization bound. To effectively use the available shared cache on multicore platforms, scheduling methods must be aware of it. Considering this problem in the context of soft real-time application, we presented an improved hybrid scheduling algorithm on multicore platform. Therefore, we proposed two new agorithms in this research work:

First, based on the EDF-US[1/2] scheduling algorithm, we presented an cluster-based hybrid scheduling algorithm. In this approach, the homogenous multicore is averagely partitioned into clusters. Based on worst fit heuristic algorithm tasks are statically assigned to these clusters after sorting the tasks by descresing order, and in each cluster the number of the heavy task which utilization is greater than 1/2 must not more than the number of the core in the cluster. Tasks are scheduled within each cluster using global EDF-US[1/2] scheduling algorithm and not allowed to migration. Our work gives the lower bound on the utilization bound of the improved algorithm. Then compared to global scheduling and partitioned scheduling, we analyses the performance of the hybrid scheduling algorithm.

Second, in order to reduce the hierarchically shared cache contention and effectively use the available shared cache, based on the classical task model we introduced a new task model considering the resourcing constraints of tasks, and then

proposed an improved hybrid approach for scheduling real-time tasks on multicore platforms. The multicore platform is divided into clusters of cores according to whether the cores share a common L2 cache, and then tasks which access the same memory region are statically assigned to the same cluster using heristic algorihm. Devising mechanisms encourage those tasks to be co-scheduled within each cluster. Our simulation results show the effectiveness of the proposed algorithm can improve the cache performance and increase the cache hit rate.

Key Words: Homogeneous Multicore Processor; Real-Time Scheduling; Hybrid Scheduling; Utilization Bounds of Agorithm; Shared Cache

插图索引

图 2.1 Intel Core Duo 双核处理器的逻辑结构 5
图 2.2 AMD 双核 Opteron 处理器的逻辑结构 5
图 2.3 多核局部调度算法调度方案 10
图 2.4 多核全局调度算法调度方案 10
图 2.5 任务迁移的通信开销 17
图 3.1 混合调度算法调度方案 19
图 3.2 全局 EDF 调度算法调度情况 20
图 3.3 EDZL 调度算法调度情况 20
图 3.4 混合调度算法调度情况 20
图 3.5 基于簇集的混合 EDF 调度算法调度情况 25
图 4.1 多核架构中多级 Cache 结构示意图 32
图 4.2 共享缓存对调度的影响 33
图 4.3 高速缓存层次结构的多核 37
图 4.4 基于共享缓存策略的混合调度算法的步骤 38
图 4.5 任务合并实例 38
图 4.6 全局 EDF 调度情况 42
图 4.7 改进 GEDF 调度情况 42
图 4.8 仿真多核结构示意图 44
图 4.9 算法失效率比较结果 46

附表索引

表 2.1 算法及其近似比 12
表 4.1 待任务调度集实例 41
表 4.2 任务各个参数的生成情况 44
表 4.3 采用不同分配算法时 Cache 失效率测试结果 45
表 4.4 簇集内采用不同调度算法时 Cache 失效率测试结果 45
表 4.5 采用不同分配算法时 Cache 失效率测试结果（cache=1M） 45
表 4.6 簇集内采用不同调度算法时 Cache 失效率测试结果（cache=1M） 45

第1章 绪论

本章概述了多核实时调度的研究背景及意义，同时，阐述了多核实时调度的研究现状。最后，介绍了本文的主要研究内容及其组织结构。

1.1 课题来源

本课题来源于国家核高基重大专项项目：实时嵌入式操作系统及开发环境（2009ZX01038-001）。

1.2 研究背景及意义

从1945年第一台计算机问世以来，计算机技术突飞猛进，使得计算机在当今的社会里变得无所不在，被应用到越来越多的系统中。在众多实时控制领域中，如汽车、飞机、武器、精密加工机床等，计算机技术得到了广泛应用。通常，用于实施控制的系统，就称为实时系统。所谓实时，是指在一个确定的时间里，对外部产生的事件做出响应，并在确定的时间里，完成这种响应及处理[1]。因此，实时系统不仅要保证事物运行的逻辑正确性，而且要保证事物运行的时间正确性。例如，用于航空航天、导弹发射等领域的实时系统，要求计算机能及时的加工处理相关的数据，否则将造成不可弥补的重大灾难。

近年来，单核处理器中的晶体管数目越来越多、频率越来越高，使得处理器的功耗急剧增加，发热问题日益凸显；另外，为了提高指令集的并行性，芯片设计越来越复杂，导致处理器结构设计和人力成本急剧上升，同时也限制了处理器并行性的发掘。综合这些问题可以得到一个简单结论：目前单个微处理器芯片集成了数量空前的晶体管，如果继续保持每年或两年完成新一代更大规模处理器的设计与调试工作，将会使成本直线上升以致无法完成设计工作[2]。因此，在单核微处理器发展遇到瓶颈后，纷纷转向片上多处理器（Chip Multiprocessor）的发展，通过并行计算来提高计算性能。近年来，处理器产商纷纷推出自己的多核结构，例如AMD公司的Opteron、Phenom，Intel公司推出的Core Duo、Core i3、i5、i7等，还有IBM公司开发的POWER4、POWER5、PowerPC970和ARM推出的MPCore等处理器， 无论是在大型机领域还是嵌入式领域，处理器都进入了多核时代。多核处理器平台应用于高端实时应用系统的增长趋势显著。在航空航天和国防领域，多核芯片的使用可以满足减小处理器尺寸、重量和功率的应用要求，因此，多核在这些领域得到越来越多的应用。在汽车电子和卫星系统领域，类似的趋势也很明显，欧洲航天局（ESA）表示支持适用的多核处理器在航天领域的应用发展[3]。

多核处理器架构不同于单核处理器架构，如何更有效的开发多核处理器架构的并行性，就给任务调度中的相关研究带来了新的变化。而且，在实时系统中，为了保证实时性能，同时有效的利用可用的处理器能力，需要高效的调度算法和准确的调度分析技术来支持实时系统的应用需求。因此，多核实时调度算法的研究对多核在实时领域的应用具有重要意义。

1.3 研究现状

单核处理器实时调度算法的理论研究可以追溯到1973年Liu和 Layland发表的文献[4]，文献中 Liu 和 Layland 首次提出了周期任务模型，并在此基础上提出了单调速率调度算法和最短截止时限优先算法。随后，一系列的单核实时调度算法研究在这篇文章的基础上开展起来。其中，1995年Audsley的研究成果[5]和2004年 Sha 的研究成果[6]促进了单核实时调度研究的发展历程。至今，仍然有许多学者在这一领域进行研究，但是，随着众多理论成果的提出以及这些算法在实际系统的成功运用，单核实时调度理论的研究已趋于成熟。

随着多核处理器的出现，对于多核实时调度算法的研究需求日趋增大，多核实时调度理论的研究同样可以追溯到上世纪六十年代末七十年代初，Liu [7]指出多核实时调度算法在本质上比单核实时调度算法要难：用于单核处理器的调度方法，不能直接推广到多核处理器上使用，只存在一种简单的情况就是一个任务只能等待一个特定的处理器对其进行处理，即使还有其他可用的空闲处理器，这就给多核实时调度带来了新的问题。同时，1978年Dhall和Liu [8]指出，对于允许任务从一个核迁移到另一个核执行的多核全局调度算法，会产生“Dhall 效应”，随后的二十几年研究学者主要集中研究多核局部调度算法。局部调度算法，通常是将任务分配算法与单核实时调度算法相结合，Dhall和Liu [8]基于单调速率调度算法提出了RMNF算法和RMFF算法。随后，Oh和Son[9]在Dhall和Liu的基础上，缩小了在最坏情况下RMNF算法和RMFF算法的近似比，提出了RMBF算法，并且，在文献[10]中进一步提出了RRM-FF算法和RRM-BF算法。Burchard等[11]1995年提出了RMGT算法。局部调度算法中的任务分配是NP难问题[3]。因此，局部算法的研究主要是提出适合的启发式分配算法，近年来，有很多学者提出采用遗传算法[12,13]或模拟退火算法[14]来进行任务分配。

1997年Phillip[15]指出全局调度算法中的“Dhall 效应”是由于高利用率的任务对算法的影响，人们开始重新关注多核全局调度算法。Anderson B 等人[16]指出可以将单核实时调度算法扩展到多核全局调度算法，并提出了RM-US[m/(3m-2)]算法和 EDF-US[m/(2m-1)]算法。Goossens 等[17]研究了多处理器平台上基于优先级驱动的全局调度算法，同时，Baker[18,19]在多处理器平台上对实时调度算法做了大量研究，并针对全局调度算法的可调度性分析做了大量的研究。Baruah[20]根据

按比例公平使用资源的思想，提出了按比例公平调度算法，PFair 算法被认为是多核调度算法中最优的全局调度算法。基于 PFair 公平调度算法，研究学者提出了一系列的改进算法如 ERfair、PD 和 PD2[21,22]。

针对全局调度和局部调度各有优缺点，近年来，许多学者提出了综合全局调度和局部调度优点的混合调度算法。美国北卡大学的 Anderson 和 Brauch 针对多核实时调度做了大量研究，提出了多种混合调度算法[23,24]。同时，Anderson 小组针对共享缓存对多核实时调度的影响，提出了多种启发式策略[25~27]。

1.4 本文研究工作

针对同构多核的广泛应用，本文对多核实时调度算法进行了研究和分析。具体的研究工作包括以下几个方面：

（1）对实时系统中的相关概念和实时调度进行了分析。分类研究了多核实时调度中的全局调度和局部调度，并针对典型的局部调度算法和全局调度算法进行深入分析。概述了多核实时调度中局部调度和全局调度的具体实现，并指出了它们各自的优缺点

（2）对于综合了局部调度和全局调度优点的混合调度算法进行了介绍，给出了相关的分析，并概述了混合调度算法的研究现状。提出了一种基于全局 EDF-US[1/2]调度算法的混合调度算法，并给出了该调度模型下混合调度算法的利用率，结合给出的利用率界限，分析了该算法与全局调度算法和局部调度算法的性能。

（3）分析了多核平台上共享高速缓存对调度过程的影响，研究了相关的改进方法。在混合调度算法的基础上，提出了一种基于高速缓存意识的混合调度算法。研究并设计了仿真实验环境，通过实验验证了算法的有效性。

1.5 本文组织结构

第一章 绪论部分

介绍了本文的研究背景及意义、多核实时调度的研究进展，以及本文的研究内容和论文结构。

第二章 多核处理器实时调度相关研究

针对同构多核处理器的特点，首先对实时系统的概念、特点以及实时任务进行了概述，并对实时调度相关概念和实时调度的分类进行了简单介绍；然后，着重介绍了多核实时调度算法的分类及其研究现状，并对其进行了分析；最后，阐述了混合调度算法的研究现状。

第三章 采用分簇的混合调度算法及其利用率分析

针对局部调度算法和全局调度算法的不足，基于全局 EDF-US[1/2]调度算法，

提出了一种混合调度算法。该算法首先将多核平台上的多个核平均分成多个处理器组，然后将任务集中的任务分配到各个处理器组，并在组内采用全局调度算法。随后，文章给出了该混合调度算法的利用率，在此利用率的基础上，分析比较了了混合调度算法相对于全局调度算法和局部调度算法的优势。

第四章 面向共享缓存改进的多核实时调度算法

针对共享高速缓存对多核调度的影响，对其进行了分析，并简要介绍了现有的解决方法。混合调度算法得到了越来越多的关注，因此，文章在混合调度算法的基础上提出了基于共享缓存策略的混合调度算法。在建立任务调度模型时，提出读取相同内存资源的任务为存在数据依赖，并在分配阶段，将有数据依赖的任务尽最大努力分配到同一个共享二级缓存的处理器分组，随后，在处理器分组内鼓励有数据依赖的任务共同调度。分析得出，算法的延迟界限是有界的，这对于软实时应用是允许的。同时，仿真测试得出，该算法有效改进了任务运行时共享二级缓存的命中率。

结论和展望

对本文的工作进行了总结，对研究的问题和今后的工作进行了展望。

第2章 多核处理器实时调度相关研究

本章主要介绍多核实时调度的研究现状，对典型多核调度算法的具体实现及其可调度性进行了简要概述。为了更好的理解多核实时调度，文章首先论述了实时系统的概念及特点，并且，对实时调度中的相关概念和分类进行了阐述。

2.1 概述

多年以来为了提升处理器的性能，设计者不断提高处理器工作频率和晶体管的数目。首先，随着晶体管数目的增多和处理器时钟频率的不断提升，处理器的功耗急剧增加，发热问题日益凸显，采用这种方法提升处理器性能已接近极限；再者，为了挖掘指令集并行以提升处理器速度，使得硬件和软件设计的复杂度日益提升，而复杂的硬件和软件设计限制了指令集的并行。在这种形式下，多核处理器（Multicore Processors），也称为片上多核处理器（chip multiprocessor, CMP）应运而生。多核处理器将多个完全功能的核心集成在同一个芯片内，整个芯片作为一个统一的结构对外提供服务，输出性能[28]。相比于单核处理器，多核处理器内集成了多个处理器核，提升了处理器的并行性，其次，依靠多个核来提升处理器性能，每个核的主频可相对低些，从而可以降低功耗[29]。由于多核处理器的高效性能，多核处理器得到了广泛应用，如图 2.1 和图 2.2 所示，处理器生产商如 Intel 和 AMD 纷纷推出了各自的多核处理器。

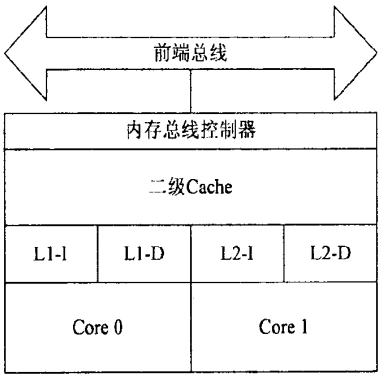

图 2.1 Intel Core Duo 双核处理器的逻辑结构

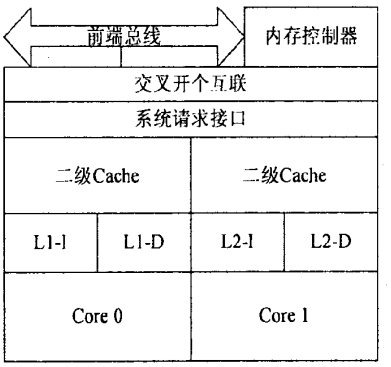

图 2.2 AMD 双核 Opteron 处理器的逻辑结构

多核处理器的组织结构没有一定的限定，根据多个处理器核对等与否，可将多核处理器系统分为两类：

（1）同构多核处理器（Homogeneous CMP）：处理器核由结构相同的通用处理器核组成，多个处理器核执行能力相同，地位对等。典型的同构多核结构有斯

坦福大学研制的多核处理器 Hydra，Hydra 包含四个相同的 MIPS 内核，每一个内核具有私有的数据和指令缓存，所有的内核通过总线共享一个统一的大型片上二级缓存进行通信；同时，现在 Intel 和 AMD 主推的双核处理器都是同构处理器。

（2）异构多核处理器（Heterogeneous CMP）：处理器中的各个核结构不近相同，各个核之间存在差异，地位不对等，执行任务的能力和特性各不相同。异构多核主要用于特定的应用，多采用“主处理器+协处理器”的设计结构。IBM、索尼和东芝等联手设计推出的 Cell 处理器正是这种异构架构的典范，Cell 处理器包含一个 PowerPC 的通用处理器核和 8 个辅助处理器核，通过一条高速总线进行连接，它主要用于高效率、低功耗地处理下一代宽带多媒体和图形应用[28]。

处理器结构的改变，给软件的设计带来了冲击，只有与多核硬件相适应的软件，才能真正的发挥多核的性能，这就对编译器、操作系统的设计都提出了要求。众所周知，对于实时系统而言，选取的调度算法将决定调度的有效性，如何更有效的开发多核处理器的并行性，就给实时系统中任务调度的相关研究带来了新的挑战。

2.2 实时系统

为了更好的研究多核实时调度算法，本节将具体介绍实时系统中的相关概念，如实时系统的定义及实时系统的特点等。

2.2.1 实时系统概念

实时系统[30]是指对于任何外部事件都能由计算机给出及时响应的系统。实时系统与其他系统的最大区别在于，其处理和控制的正确性不仅取决于技术的逻辑结果，而且取决于计算机和处理结果需要的时间。因此，对于一个实时系统而言，就是要尽最大努力确保任务的时限得到满足。

2.2.2 实时系统特点与分类

对于实时系统而言，其特点主要有以下几点[1]：

第一，任务具有截止期。实时系统在响应外部事件时，该外部事件我们称之为任务，必须在一个给定的时间内完成，这个给定的时间即是任务的截止时限。根据任务的截止时限，实时系统又可以分为硬实时系统和软实时系统。

第二，实时任务具有结构上的内在联系。实时系统中的各个任务之间为了完成某个动作，在结构上相互关联，成为一体。

第三，实时系统中，任务的活动具有各种要求和限制，如计算时间要求、资源要求、通信上的要求、执行次数的限制、执行时间的限制等。

第四，实时任务是可预测的，实时系统要求事先知道任务在最坏情况下的执行时间以及所需的数据和资源，并且对最坏执行时间的预测尽可能接近实际运行

情况。根据实时任务的可预测性，实时系统可知所有任务的运行是否满足时间限制。

第五，实时系统中的所有实时任务，都是由外部事件激发的。

第六，在实时系统中，任务的活动是不可逆的。

2.2.3 实时任务

实时任务根据不同的划分标准，有不同的划分结果，主要分为以下两种划分标准。

（1）根据任务是否周期性释放，可将任务分为周期任务（Periodic Task）和非周期任务[30]：

周期任务：实时系统中以固定时间间隔周期释放并执行的任务，就称为周期任务。因此，一个周期任务可以被看作包含多个作业，每个作业以固定间隔周期性释放。实时系统中有很多任务都是周期任务，例如，一个任务可能希望每 100*ms* 检测一次飞行器的速度、高度和空间方位角，以获得飞行器的控制信息。

非周期任务：非周期任务执行的间隔时间是不确定的，如移动通信设备中的通信任务，该任务只有在需要通信时才会执行。非周期任务又可以分为两种情况：有最小到达间隔时间限制的非周期任务（Sporadic Task），这种任务又称为零星任务；另一种就是没有到达时间限制的非周期任务（Aperiodic Task）。

（2）根据对任务的截止时限（Deadlines）要求，实时系统中处理的任务可以分为硬实时任务（Hard Real Time Task）和软实时任务（Soft Real Time Task）：

硬实时任务：要求任务必须完全满足任务的时限要求，对于硬实时任务而言，任务完成了但是超过了截止时限是没有任何意义的，甚至可能带来巨大的影响。

软实时任务：允许系统对任务的截止时限有一定的延迟，其截止时限要求只是一个相对条件。虽然任务的响应和执行超过了截止期，但是，这种响应可能还有一定的价值[31]。

2.3 实时调度

本节主要介绍实时调度中的相关概念和定义，以及实时调度的分类。

2.3.1 实时调度相关定义

针对不同的多核处理器结构，任务的执行情况也有所不同：同构多核处理器结构中每个核有相同的执行能力，因此，在不同的处理器核上运行的任务，处理器核对其执行速率都是相同的。但是，异构多核处理器核的结构不尽相同，处理任务的能力也不相同，因此，一个任务的执行速率不仅取决于任务本身，还和所在的处理器核有关，并且，并不是每个核上都可以运行所有的任务。本文主要研究同构多核上的实时调度问题。

多核实时调度的研究主要集中于两个任务模型：周期任务模型和零星任务模型。周期任务模型是 Liu.C 和 Layland.J 在 1976 年提出，提出的周期任务集要求如下：

（1）所有任务的截止时限是周期性的，有固定的时间间隔；

（2）每个任务必须在下一个任务请求发生前完成；

（3）任务是相互独立的，不依赖于任务的初始状态和其他任务；

（4）每个任务的运行时间是恒定的，运行时间是指任务不被中断的情况下处理器执行该任务所需的时间；

（5）每个非周期任务在系统中都是特例。

在周期任务模型中，每个任务中的作业以固定的时间间隔周期性释放；在零星任务模型中，每个任务可在任何时间到达，即任务中一个任务的释放与下一个任务释放时间间隔不固定。周期任务集又可以分为同步周期任务集和异步周期任务集，同步是指所有的任务在同一个时间点同时到达，异步是指任务的到达时间是由固定负载分开的，没有一个相同的到达时间。在零星任务模型中，任务的到达时间是相互独立的。同一个任务中的多个作业不允许并行执行，而且，在任意时刻，每个作业只能在一个处理器上运行。

基于周期任务集，每一个任务t_i都可以描述为$t_i = (T_i, C_i, D_i)$，其中D_i表示任务的相对截止时限，C_i表示任务的最差执行时间，T_i表示间隔的最小到达时间或周期，U_i表示任务的利用率，$U_i = C_i / T_i$。对于含有 n 个任务的任务集，U_{SUM}（利用率总和）表示所有任务的利用率的和，其定义如公式（2.1）所示：

$$U_{SUM} = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.1}$$

对于周期实时任务，根据任务截止时间可以分为三类：

（1）Implicit deadlines:所有任务的截止时间等于他们的周期，即$D_i = T_i$；

（2）Constrained deadlines;所有任务的截止时间都小于或等于他们的周期，即$D_i \le T_i$。

（3）arbitrary deadlines;任务的截止时间可能小于，等于或大于他们的周期。

2.3.2 实时调度分类

调度就是将计算机中的资源分配给任务（或给任务分配资源）。从不同的角度看，实时调度有以下几种分类方法：

（1）静态表驱动调度和优先级驱动调度

按照调度算法中决定任务调度的时刻可以分为静态表驱动调度和优先级驱动调度。

静态表驱动调度：在系统开始执行前制定一张时间表，此时，系统中的每个任务何时开始执行和结束时间都是确定的。时间表一旦形成，任务的执行顺序就

是确定的，不会发生改变，因此，该方法又称为静态调度。

优先级驱动调度：为每个任务分配一个优先级，并总是选择最高优先级的任务执行。优先级驱动调度最重要的就是确定任务之间的相对调度顺序，即哪些任务相对其他任务拥有更高的优先级，哪些任务的优先级较低。优先级驱动调度算法又可分为静态优先级调度和动态优先级调度。相对于静态表驱动调度，优先级驱动的调度算法更能满足时限要求。例如，一个之前不知道的任务要求执行，由于静态调度事先并不知晓这个任务的存在，所以该任务将不会得到执行。

对于优先级驱动调度算法，给任务分配的优先级将决定任务运行过程中其截止时限能否得到满足，其中一个方法就是给最重要的任务分配最高的优先级，那么当有任务错过截止时限时，对系统的总体影响也不是很大。但是，这个方法的缺点是可能存在另外一个调度策略可以满足所有任务的截止时限要求。为了确定是否会有任务错过他们的截止时限，需要进行可调度性分析。

（2）抢占式调度和非抢占式调度

按照执行过程中任务运行时是否允许被抢占，又可以分为抢占式调度和非抢占式调度。

抢占式调度：在任意时刻，当正在运行的任务比其他任务的优先级低时，拥有更高优先级的任务可以抢占处理器执行。抢占式调度算法可优先满足高优先级任务的截止时限要求，实时性能更好。但抢占式调度算法的上下文切换次数高，导致这一部分的开销很大。

非抢占式调度：一旦一个任务开始在处理器核上执行，直到任务执行完，除非任务自动放弃执行，任务都不能被其他任务中断，即使其他任务拥有更高的优先级。非抢占式调度算法的上下文切换次数低，开销小，但是，在实时性方面，没有抢占式调度性能好。

（3）单核调度和多核调度

按照处理器的类型，实时调度还可以分为单核处理器调度和多核处理器调度。

单核处理器调度：调度算法只需考虑使用哪种方法调度任务，而无需考虑如何将任务分配到各个处理器核，因为硬件结构就只有一个核。

多核处理器调度：随着多核处理器架构的出现而出现的多核处理器调度，可以被视为需要尝试解决两类问题：第一，分配问题，即任务应该在哪个核上执行；第二，优先级问题，即任务何时执行以及任务与其他任务之间的执行顺序。

2.4 多核实时调度算法

多核调度主要分为局部调度和全局调度。局部调度的主要思想是将任务按照一定的规则分配给各个处理器核，每个处理器核维护本地的调度队列。在不考虑分配到不同的处理器核上的任务相互之间的联系的情况下，可以认为处理器核上

的任务独立于其他核上的任务。因此，多核处理器上的任务调度问题就转化为了单核处理器上的调度问题。一旦任务分配完成后，大量用于单核的实时调度算法可以被使用。多核局部调度方案如图 2.3 所示：

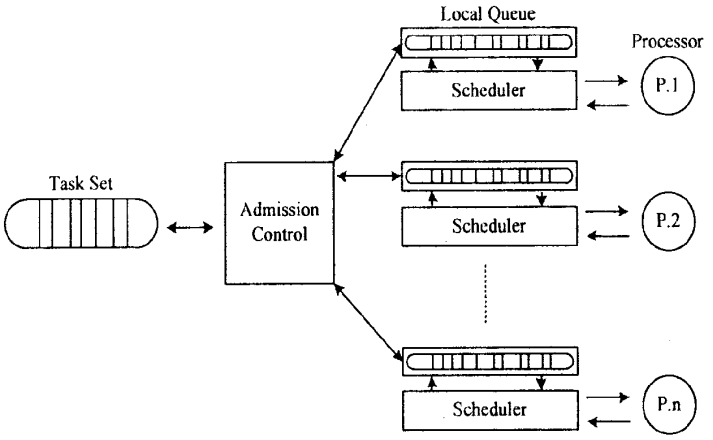

图 2.3 多核局部调度算法调度方案

全局调度中将所有任务交给一个调度队列，从中选择合适的任务到处理器核进行调度。全局调度算法允许任务在核间迁移和抢占执行。多核全局调度方案如图 2.4 所示：

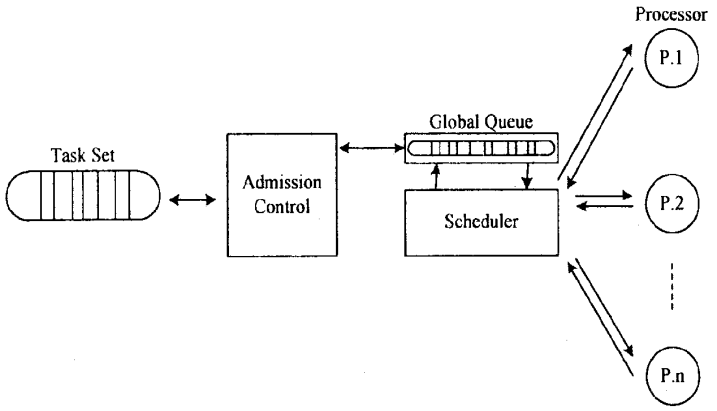

图 2.4 多核全局调度算法调度方案

2.4.1 局部调度

局部调度是将任务静态的分配到各个处理器核，任务只允许在分配的核上运行。因此多核调度问题就转化为了多个单核调度问题。局部调度主要分为两步：任务分配和单核处理器上的任务调度。

单核上的任务调度经过多年的发展，已经有一套较成熟的调度理论。以下的单核处理器调度算法对多核局部调度有很大的影响，它们分别是：单调速率[4]（RM，Rate Monotonic）算法、最短截止时限优先[4]（EDF，Earliest Deadline First）算法、单调期限[32]（DM，Deadline Monotonic）调度算法和最小空闲时间优先[33]

（LST，Least Laxity First）算法。

任务分配问题类似于装箱问题，有多种启发式算法，如首次适应（FF，First-Fit）算法，最佳适应 （BF，Best-Fit）算法，最差适应 （WF，Worst-Fit）算法和下次适应（NF，Next-Fit）[34]算法，同时，研究者提出先将任务按照利用率递减（DU，Decreasing Utilisation）或递增进行排序再结合 FF、BF、NF、WF 算法进行分配的方法。

FF 算法：算法首先扫描待调度的任务，依次将任务队列中的任务分配给一个最近的可接受它的非空处理器核，可接受即待调度任务的利用率不超过处理器核所能调度的能力。FF 算法只有在当前已被分配任务的核都不适合该任务时才给任务分配空核。

NF 算法：算法同样先扫描待调度的任务，将任务分配给处理器核后，若下一个待调度的任务不能分配到当前处理器核，就将该任务分配给下一个未分配任务的处理器核，而不检查前面是否有可接受该任务的处理器核。

BF 算法：BF 算法类似于 FF 算法，不同之处是，算法在可接受该任务的处理器核中选取剩余利用率最小的处理器核，将任务分配给该处理器核。

WF 算法：算法在可接受当前任务的处理器核中选取剩余利用率最多的处理器核，将任务分配给该处理器核。

其中，WF 算法和 NF 算法将任务均衡的分配到各个核，使得分配较均衡；而 FF 算法和 BF 算法使每个核分配尽量多的任务，从而可能使得其他的处理器核空转，造成整个系统的不平衡。由于多核处理器任务分配问题是 NP 难问题，现在有很多研究使用模拟退火算法、遗传算法进行多核任务分配。

算法给任务分配的优先级是固定不变的，即优先级在设计时就确定了，且在运行时优先级不会改变，则称这个调度算法为静态的，静态调度算法又被称为优先级固定算法，如果任务的优先级是不断变化的，则调度算法是动态的。RM 调度算法和 DM 调度算法都是固定优先级调度算法，EDF 调度算法和 LST 调度算法是动态优先级调度算法。

RM 调度算法由 Liu.C 和 Layland.J 在 1976 年提出，该算法基于任务的周期和任务的截止时限确定优先级，算法假定任务的相对截止时限等于任务的的周期，任务的周期越短，截止时限越紧迫，优先级越高。如果任务的总利用率满足公式（2.2），算法可以在满足所有任务的截止时限下产生可调度的队列。

$$U_{SUM} \le n(2^{1/n} - 1) \tag{2.2}$$

其中，n 为任务的个数。当$n \to \infty$，$U \approx \ln 2$，由此可知，使用 RM 算法进行调度可以使处理器的利用率达到 70%以及更高。但是，这是充分非必要条件，使用 RM 调度算法时，可以调度总使用率大于$n(2^{1/n} - 1)$的任务。

单调期限（DM）调度算法同单调速率（RM）算法一样是一种常用的固定优

先级调度算法，DM 调度算法是基于 RM 调度算法进行改进的一种硬实时调度算法。它适用于 constrained deadlines 任务集，即任务的截止时限小于任务的周期。因此算法按照任务的截止时限来分配优先级：截止期时越短，优先级越高。即时限宽度越短，优先级越高，时限宽度越长，优先级越低。

EDF 调度算法根据任务截止时限值分配任务优先级，是一种动态的优先级调度算法。对于一个任务，任务的截止时限越近，任务的优先级越高，相反，任务的截止时限越远，则任务的优先级越低。使用 EDF 算法进行调度可以使处理器的利用率达到 100%。EDF 调度算法是一个最佳的单处理器调度算法，即如果一个任务集不能使用 EDF 调度算法调度，那么其他的调度算法也不能调度。

Mok 提出的最小空闲时间优先（LST，Least Slace-Time First）算法与后来 Michael 提出的最小裕度时间优先[35]（LLF，Least Laxity First）算法的原理是相同的，都是按照任务执行的缓急程度来给任务分配优先级的算法。

LLF 算法给每个任务计算任务的富余时间，假设在 t 时刻，作业 j_i 富余时间定义为 $l(t)$，其中作业在时刻 t 剩余部分的执行时间为 e_i，绝对时限时限为 d_i，那么作业在 t 时刻的富余时间 $l(t) = d_i - t - e_i$。富余时间被称为裕度，裕度小的，优先级越高，也就需要更快的执行。LLF 算法为动态优先级调度算法，在每个时钟中断时，都要计算新加入任务和系统原有任务的富余时间，由于正在运行的任务富余时间不变，而等待被调度的任务富余时间越来越少，所以优先级发生改变。但是，所有的任务随着时间向前推进，所剩的富余时间越来越少，可能引发频繁的任务抢占，导致 LLF 算法执行时系统开销很大，因此，LLF 算法的实用价值不高。

表 2.1 算法及其近似比

算法	近似比
RMNF	2.67
RMFF	2.33
RMBF	2.33
RRM-FF	2
FFDUF	2
RMST	7/4
RMGT	1/（1-u_{max}）
EDF-FF	1.7
EDF-BF	1.7

对于局部调度算法，通常是将启发式分配算法与单核实时调度算法相结合，Dhall 和 Liu 在文献[8]中提出了 RMNF 算法，随后，Oh 和 Son[9]基于 LLWC 提出了 RMFF 算法和 RMBF 算法。Burchard 等[10]提出了 RMGT 算法。表 2.1 中给出了几种常见的局部调度算法及其近似比。为了比较不同分配算法的性能，使用近

似比率$R_A = \lim_{N_o \to \infty} \frac{N}{N_o}$来衡量一个任务分配算法 A 的渐进性能界限，其中 N 表示该分配算法所需处理器数量，N_o表示最优分配算法所需处理器数量。虽然近似比率可以用于不同算法之间的比较，但是作为可调度性测试是有限的，因为最优分配所需的处理器数量求解是一个 NP 难问题。而且，近似比是在处理器数量趋于无穷时求得的。

2.4.2 全局调度

在全局调度中，调度算法能否产生可行的调度，使任务运行满足它们的截止时限，取决于给任务分配的优先级是否合适。一个简单的方法就是使用单核中的实时调度算法来分配任务的优先级，如 RM 调度算法和 EDF 调度算法。但是在多处理器中，无论是按照 RM 调度算法还是 EDF 调度算法分配优先级，系统都无法得到最优的表现性能。例如，考虑在有 m 个核的多处理器中调度 m+1 个周期任务。假设任务$\tau_i(1 \le i \le m)$的周期$T_i = 1$，执行时间$C_i = 2\varepsilon$，而任务τ_{m+1}的周期为$T_{m+1} = 1 + \varepsilon$，执行时间$C_{m+1} = 1$；所有的任务都在 t=0 时刻到达。当我们使用 RM 调度算法或 EDF 调度算法调度该任务集时，任务$\tau_i(1 \le i \le m)$在到达后会立刻执行 2ε 个时刻，随后τ_{m+1}从时刻 2ε 开始执行直到时刻 1+ε，此时，任务执行了 1-ε 个时刻，它将错过它的截止时限。该任务集的任务利用率为 U，如式（2.3）所示：

$$U = 2m\varepsilon + \frac{1}{1+\varepsilon} \tag{2.3}$$

令$m \to \infty$，$\varepsilon \to 0$，任务集的利用率趋于 1；但是，对于含有 m 个核的多处理器系统，该任务集中依然有任务错过截止时限，证明算法的性能不高。但是，如果给周期为 1+ε 的任务分配最高的优先级，那么该任务集可被调度。该现象被称为“Dhall Effect”[8]，Dhall 效应指出在多核上调度一组任务集，RM 调度算法在固定优先级调度中不是最优的，并且，虽然 EDF 调度算法在单核上是最优的调度算法，但是到多核系统中，EDF 调度算法是非最优的。随后，Dhall 和 Liu 指出将任务集中的任务分成两类，繁重任务（Heavy task）和轻量任务（Light task），繁重任务指任务执行时间与任务截止时间比值高的任务，轻量任务指任务执行时间与任务截止时间比值相对低的任务。将繁重任务与轻量任务分开对待，可有效的改进 EDF 调度算法的调度性能。

Anderson 等人[36]基于周期任务集模型提出了全局的静态调度算法 RM-US[m/(3m-2)]，该算法指出对于周期任务系统 τ，如果所有任务满足以下属性：

属性 1：对于每个任务$\tau_i \in \tau$，$u_i \le m/(3m-2)$

属性 2：$U(\tau) \le m^2/(3m-2)$

该任务集被称为轻系统。对于轻任务集，使用全局 RM 调度算法可以使所有任务截止时限得到满足。随后，作者在文章中扩展了该算法，并指出当任务利用率$u_i > m/(3m-2)$时，给任务赋予最高的优先级；如果任务利用率满足

$u_i \le m/(3m-2)$，则根据 RM 算法赋予任务优先级。那么当系统中所有任务的利用率满足$U(\tau) \le m^2/(3m-2)$，该系统使用 RM 调度算法可调度，即系统中的每个任务不会错过截止时限。

除此之外，文章在 EDF 调度算法的基础上提出了 EDF-US[m/(2m-1)]算法，算法指出如果任务 τ 的利用率$u_i > m/(2m-1)$，那么我们称这个任务为繁重任务，如果$u_i \le m/(2m-1)$，则称为轻任务。使用全局 EDF 调度算法调度一个周期任务集 τ，如果该任务集满足以下属性：

属性 1：对于每个任务$\tau_i \in \tau$，$u_i \le m/(2m-1)$，$1 \le i \le n$

属性 2：$U(\tau) \le m^2/(2m-1)$

该任务集被称为轻系统，对于一个给定的任务集，若其是轻系统，则其使用 EDF 调度算法是可调度的。EDF-US[m/(2m-1)]算法的利用率上界为$m^2/(2m-1)$，当处理器的个数很多，该算法的界限趋近与 1/2。

Srinivansan 和 Baruah 放宽了 EDF-US[m/(2m-1)]算法中对于属性 1 的限制，提出了 EDF-US[ζ]算法[37]，算法思想是给利用率超过阈值 ζ 的任务赋予最高的优先级，令阈值 ζ=m/(2m-1)，对于一个给定的任务集 τ，如果$u_i > m/(2m-1)$，那么任务被赋予最高的优先级，拥有最高优先级的任务，它们之间的优先级是任意的；如果$u_i \le m/(2m-1)$，任务τ_i按照 EDF 调度算法分配优先级。算法的利用率界限如下公式（2.4）所示，当$m \to \infty$时，利用率上界趋近于 m/2。

$$U_{EDF-US[\zeta]} = \frac{m^2}{2m-1} \tag{2.4}$$

Goossens 等人[38]给出了全局抢占式 EDF 调度算法的可调度条件，指出对于 implicit-deadline 的周期任务集，其可调度界限是：

$$U_{EDF} = m(1-\alpha) - \alpha \tag{2.5}$$

其中$\alpha = max\{u_i, i = 1,2,\ldots,n\}$。同时，文章中提出了 EDF（$k$）算法，算法指出，对系统 τ 中的 n 个任务按利用率递减排序，即$u_i \ge u_{i+1}(0 < i < n)$，如果 i<k，给任务赋予最高的优先级，若 i≥k,按照 EDF 调度算法给任务赋优先级。该算法的可调度性条件如公式（2.6）所示：

$$m \ge (k-1) + \left\lceil \frac{u_{sum} - u_k}{1-u_k} \right\rceil \tag{2.6}$$

其中u_k表示 k 个任务的利用率。

Baruah 提出的固定优先级调度算法 fpEDF[39]，fpEDF 调度算法给利用率大于 1/2 的任务赋予最高的优先级，其他任务根据初始状态按照 EDF 调度算法分配优先级，优先级一旦分配就不改变。在包含 m 个核的多核架构中，任务集的利用率满足$\sum_{i=1}^{n} u_i \le (m+1)/2$，使用 fpEDF 调度算法是可调度的。最后，作者指出该算法在固定优先级的全局调度算法中是最优的，并且当任务集的总利用率满足

$\sum_{i=1}^{n} u_i \le \max(m-(m-1)\alpha, \frac{m}{2}+\alpha)$时，使用 fpEDF 调度算法是可调度的，其中 m 为处理器核数，α 为任务集中利用率最高的任务其利用率指。

Baker 等人[19]在 EDF-US[ζ]算法的基础上提出改进算法，令 ζ=1/2,调度算法的利用率上界为公式（2.7）所示：

$$U_{EDF-US[1/2]} = \frac{m+1}{2} \tag{2.7}$$

根据按比例公平使用资源的思想，Baruah[20]提出了按比例公平调度算法（PFair, proportionate fair algorithm），PFair 算法被认为是多核调度算法中最优的全局调度算法。将时间段分割成相等长度的时间槽，对于每个周期任务 $x(x.p,x.e)$，其权值为 $x.w=x.e/x.p$ $(0<x.w<1)$，分为 $x.e$ 个子任务，调度算法在每个整数时刻 t 选择子任务到核上执行一个时间槽 $[t, t+1)$ 单位。任务 x 中的每个子任务 j 的时间参数为：最早释放时间 $r(x, j)$、最晚截止时限 $d(x, j)$、运行窗口 $w(x, j)$，其中 $r(x,j)=\lfloor j/x.w \rfloor$表示子任务的最早调度时间，$d(T_j)=\lceil (j+1)/x.w \rceil - 1$表示任务的最晚调度时间，子任务 $x.j$ 必须在相对应的窗口内执行，所有子任务按优先级组成全局队列等待调度。一个任务的不同子任务，允许在不同处理器上执行（即允许在处理器之间迁移任务），但是任一时刻一个子任务不能分配两个资源（即禁止子任务在不同的处理器核上并行）。在有 m 个核的多核上使用 PFair 调度算法调度任务，算法的利用率如公式（2.8）所示：

$$U_{PFAIR} = m \tag{2.8}$$

在实际的应用中，公平调度算法做调度决策的开销非常大，在做调度决定时所有的处理器必须同步。

基于 PFair 公平调度算法，提出了多种 Pfair 公平调度算法的改进算法：ERfair、PD 和 PD^2。Anderson 在 PFair 调度理论的基础上，提出了 ERfair[21]（Early Release fair）调度算法。ERfair 算法允许一个子任务执行完成后，与该子任务属于同一任务的后继子任务提前释放，而不必等到其运行窗口到来。PD、PD^2[22]调度算法也都是基于 Pfair 调度算法改进而来的，它们都以子任务的截止时限来确定任务的优先级。但是当子任务时限相同时，算法确定子任务的优先级方式不同。PD^2 算法在 PD 算法的基础引入了组时限的概念， 即当子任务的截止时限相同时，就根据子任务所在任务组的截止时限来确定子任务之间的优先级。若所在任务组的截止时限也相同，则认为这两个子任务的优先级相同，在调度时可以任选一个进行调度。Anderson 扩展 PFair 调度算法使其能够调度零星任务集，提出了 EPDF[40]调度算法，但是该算法只能使用策略，并不能得到最佳调度。

Lee[41]在 1994 年提出了最早截止时限零松弛（the Earliest Deadline until Zero Laxity, EDZL）调度算法。EDZL 调度算法是一个作业级动态调度算法，它综合了 EDF 调度算法和 LLF 调度算法。EDZL 调度首先根据 EDF 调度算法分配任务的优

先级，当任务的松弛度变为零时，将作业的优先级提到最高进行调度。文章指出对于两个核的处理器在线调度算法，该算法是次优的。

对于零星任务系统，在包含 m 个核的同构多核系统上使用 EDZL 调度算法调度 m+1 个任务，当任务集满足以下条件[42]时：

$$\sum_{\forall i \neq k} \beta_k^i \geq m(1-\lambda_k) \tag{2.9}$$

其中，$\lambda_k = C_i/\Delta_k$，$\Delta_k = \min(D_i, T_i)$；并且

$$\beta_k^i = \frac{n_i C_i + \min(C_i, \max(0, \Delta_k - n_i T_i))}{\Delta_k} \tag{2.10}$$

该任务集使用 EDZL 算法是可调度的。

2.4.3 局部调度和全局调度比较分析

针对不同的情况，多核调度中的全局调度和局部调度各有优势和不足，因此，不能片面的说哪种方式更优。局部调度的优势包括以下几点：

首先，每个核只需维护本地的调度队列，不像全局调度那样所有的任务共用一个调度队列，对于大系统而言，随着任务的增多，在全局调度中维护一个全局的调度队列所需的开销会变得过于繁重；

其次，当一个任务超过了它的最后截止时限，只会影响与它在同一个核上的其他任务；

再者，任何任务一旦被分配后就只能在一个核上运行，没有额外的迁移代价；

最后，在实际的应用中，局部调度的最大优点即是在任务完成分配后，有大量用于单核实时调度的调度算法可供使用。

但是，局部调度有缺点：第一，任务分配问题是 NP 难问题，第二，系统中有一些任务只有在不分配的情况下才能进行调度，第三，可用核的处理能力被分割了，例如，分配算法将一个利用率很高的任务分配到处理器核后，该处理器核所剩余的利用能力有可能永远得不到再利用，因此多核处理器的所有处理能力并没有完全被使用，整个多核架构的处理能力只能利用 50%。

相比局部调度，全局调度的优势有：首先，只有在没有空闲核的时候，任务才会抢占执行；其次，闲置的生产能力，任务提前完成匀出来的核的执行能力可被其他所有的任务利用而不限于同一个核上的其他任务；再者，如果有任务超过了他的最差执行时间，相比局部调度算法，导致整个系统的截止时限超时的可能性更低；最后，全局调度更适合开放系统，因为当任务集发生改变时不需考虑运行负载平衡或任务分配算法。

在 1978 年 dhall 和 liu 发表的文章中指出全局 EDF 有 Dhall effect，认为多核中全局调度比局部调度差，因此，在八十年代和九十年代早期，人们的研究主要集中在局部调度。除此之外，全局调度的开销很高，因为一个任务可以从一个核

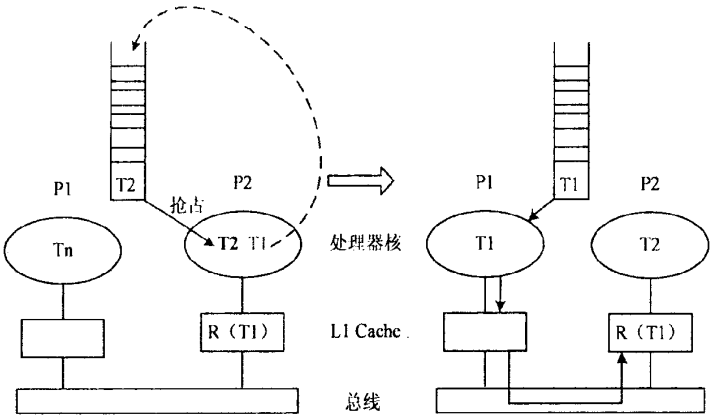

图 2.5 任务迁移的通信开销

转移到另一个核，从而导致额外的通信负载和高速缓存失效，使得最差执行时间增加，例如图 2.5 所示，任务 T_1 在一个核上运行，后来优先级更高的任务 T_2 抢占 T_1 执行，任务 T_1 转到就绪队列，等到任务 T_1 被调度的条件满足时，任务 T_1 可能在另一个核上被调度，而另一个核上没有任务 T_1 在第一个核上运行所保留的相关信息，这就导致额外的通信开销和高速缓存不命中。这些情况不会发生在完全局部调度或没有迁移的情况下。

2.4.4 混合调度

全局调度依赖于硬件结构开销很高，局部调度不能充分利用核的处理能力，因此，近几年学者们开始研究混合调度方法，混合调度方法打破局部调度中不允许任务迁移的要求，将两个或多个处理器核作为一组进行任务分配，然后在组内执行任务。这就使混合调度方法结合了局部调度算法和全局调度算法的优点。

现有的混合调度主要分为半局部调度（Semi-Partitioned Approaches）方法和分簇调度方法（Clustering）[3]。半局部调度算法的主要思想是将部分任务进行分割使其在不同的核上运行，从而充分利用在局部调度中未被完全使用的处理器核的处理能力。

Anderson B 等[23]提出了 EKG（EDF with task splitting and k processors in a group）调度算法，算法的主要思想是：基于局部 EDF 调度算法，在进行任务分配时前，将任务按照利用率从高到底进行排列，并把任务分成轻量任务和繁重任务。接着依次分配繁重任务到各个核，然后分配轻量任务，当轻量任务不能被分配时，将该轻量任务分割成两个子任务τ'和τ''，子任务τ'分配到核 p，子任务τ''分配到 p+1。算法分配部分类似于下次适合（NF）算法，但是它的不同之处是它允许将任务进行分割。被分割的子任务不允许同时执行，算法按照它们所占的利用率比例将其安排在开始和结束时的固定时间槽运行，并拥有最高的优先级，这保证任务不会并行执行。论文指出 EKG 算法的利用率取决于参数 k，参数 k 用来控制将任务分成轻量任务组和繁重任务组，EKG 算法的利用率界限为公式（2.11）

所示：

$$U_{EKG} = \begin{cases} k/(k+1) & k < m \\ 1 & k \end{cases} \tag{2.11}$$

因此，当 $k=m$ 时，利用率界限达可趋向于 100%，此时超周期内抢占任务的平均值是 $2k$。

随后，Andersson B. 和 Bletsas [24]基于将任务进行分割的思想，针对在多核上调度零星任务集提出了 EDF-SS 算法，该算法的利用率如公式（2.12）所示：

$$U = 4(\sqrt{\delta(\delta+1)} - \delta) - 1 \tag{2.12}$$

其中 δ 有效的定义时间槽长为$(\delta/minT)$。

分簇调度算法可以看做是特殊的局部调度算法，算法不是将任务分配到各个处理器核，而是首先对多核中的核进行分组，令一个分组内的处理器核为一个处理器簇集，然后将任务分配到各个处理器簇集，再在簇集内部运用全局调度算法。基于簇集的混合调度又可以分为两类：物理簇集和虚拟簇集，它们之间的区别在于处理器核映射到簇集的方式，物理簇集是指多核中的处理器核静态的映射到簇集，而虚拟簇集是指处理器核动态的映射到簇集。基于物理簇集的混合多核调度算法，当任务静态地分配到各个簇集后，只有簇集内的调度，簇集之间没有干扰。而对于基于虚拟簇集的混合调度算法，不仅有簇集内调度，还有任务在簇集间的调度。

Shin 和 Easwaran 等人[43]提出了一种基于虚拟簇集的分层调度算法，为了确定算法的利用率，算法引入了多核周期资源（MPR，multiprocessor periodic processor）模型代表处理器簇集，并将任务集分成多个簇集，确定各个任务集簇集所需的 MPR，从而得到了算法的利用率界限。

Calandrino[44]等人针对基于物理簇集的混合调度算法作了研究，在大规模的共享分层高速缓存多核平台上，提出了一种混合调度算法。算法提出将共享高速缓存的多个核作为一个簇集，并着重研究了一个物理簇集中适合的处理器核数的个数。

2.5 小结

本章主要介绍了实时系统的概念及其特点，为了后面研究的需要，介绍了实时调度中相关概念和实时调度的分类。同时，综述了多核实时调度算法的分类：局部调度和全局调度。对多核实时调度的现状进行了描述，阐述了多种实时调度算法。简要介绍了本文所述混合调度的概念和相关研究现状。

第3章 采用分簇的混合调度算法及其利用率分析

现有的多核实时调度算法主要分为两种类型：局部调度和全局调度。局部调度只允许任务在预先分配的核上运行，不能充分的利用多核平台中处理器核的剩余执行能力。全局调度算法允许任务在不同的核间动态地迁移执行，但频繁的迁移会导致额外的通信开销和缓存失效。因此，结合了全局调度和局部调度优点的混合调度算法得到越来越多的关注。本章针对全局 EDF-US[1/2]调度算法，提出了一种混合调度算法，并重点分析了该混合调度算法的可调度性，给出了改进算法的利用率下界。此外，将本文提出的混合调度算法与全局调度算法和局部调度算法进行了比较分析。

3.1 分簇混合调度算法研究

在上一章详述了混合调度算法的分类后，本章考虑分簇调度中基于物理簇集的混合调度算法。基于物理簇集的调度算法首先将多核平台中的多个核进行分组，令同一个分组中的多个处理器为一个处理器簇集，每个簇集都作为一个独立的调度系统。然后将任务静态的分配到各个处理器簇集，并在簇集内采用全局调度方法。基于簇集的混合调度算法调度方案如图 3.1 所示，可知，分簇混合调度算法主要分为两步：任务分配和簇集内的全局调度。

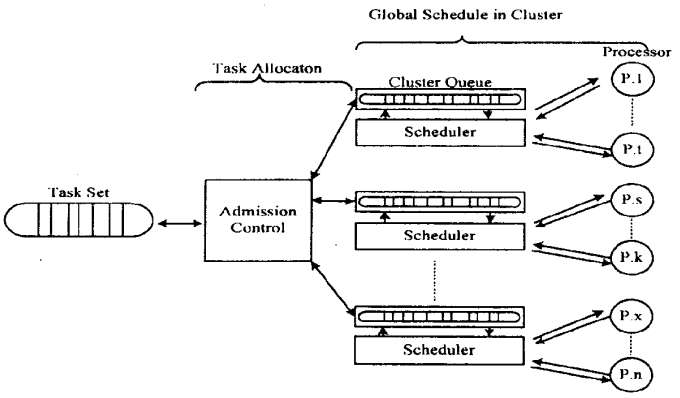

图 3.1 混合调度算法调度方案

不同于局部调度算法，混合调度算法中每个簇集包含有两个或两个以上的处理器核，当任务分配算法将利用率很高的任务分配到一个处理器核后，该处理器核剩余的处理能力可与同一簇集内的其他处理器核所剩余的处理能力合并，因此，混合调度算法中处理器核的处理能力可得到更充分的利用；前文说到，全局调度算法只有一个调度队列，当系统规模很大时，任务数的增多会使全局调度队列的

开销急剧增大。对比全局调度，分簇调度算法先将任务分配到各个处理器簇集，再在簇集内进行全局调度，对于一个大系统而言，算法运行有多个调度队列而不只有一个调度队列，调度的开销不会像全局调度算法那样急剧增大。

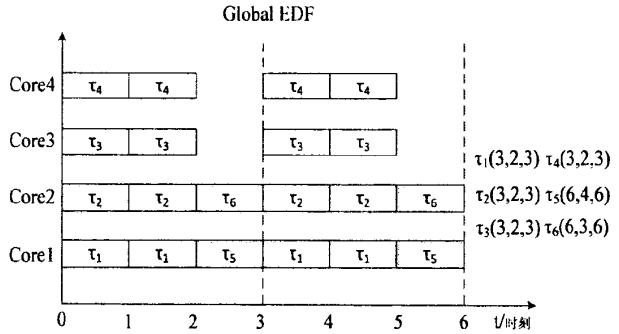

图 3.2 全局 EDF 调度算法调度情况

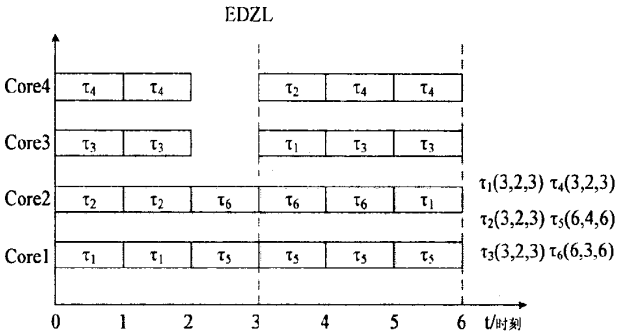

图 3.3 EDZL 调度算法调度情况

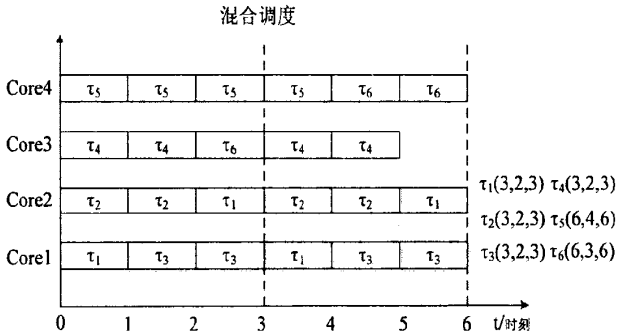

图 3.4 混合调度算法调度情况

为了更好的解释分簇调度算法以及其相对全局调度算法和局部调度算法的与优势，通过如下例子进行说明。考虑一个包含 6 个任务的零星任务集：$\tau_1 = \tau_2 = \tau_3 = \tau_4 = (3,2,3)$， $\tau_5 = (6,4,6)$和 $\tau_6 = (6,3,6)$。任务 τ=(T,C,D)，T 表示任务的周期，C 表示任务的执行时间，D 表示任务的相对截止时限。在包含 4 个处理器核的多核架构中调度该任务集，可以看出，局部调度算法不能调度该任务集，因为每个核被分配一个任务后，没有足够的容量容纳其他剩余的任务，否则将使处理中容纳的任务其总利用率大于 1，超过处理器的处理能力。使用全局 EDF，EDZL，LLF 调度算法也不可以调度该任务集。使用全局 EDF 调度算法调

度该任务集的情况如图 3.2 所示，图中横坐标表示时间，纵坐标表示任务所在的核，可知，任务τ_5、τ_6都将错过截止时限。使用全局 EDZL 调度算法调度该任务集如图 3.3 所示，情况较全局 EDF 调度好，但是任务τ_2仍将错过截止时限。考虑使用混合调度算法调度该任务集，将核 1、核 2 作为一个簇集 C_1，核 3 和核 4 为一个簇集 C_2，任务τ_1、τ_2、τ_3分配到第一个簇集 C_1，τ_4、τ_5、τ_6分配到第二个簇集 C_2，组内使用全局 EDZL 调度算法进行调度，调度情况如图 3.4 所示，该混合调度算法可调度该任务集，所有任务均在截止时限前完成。由此可知，混合调度算法在一定条件下，综合了局部调度和全局调度算法的优点，使得系统的可调度上限性能提高了。

3.2 任务模型

本文基于基础周期任务模型进行调度研究，在有 m 个相同处理能力的多核处理器架构上，调度包含有 n 个周期任务的任务集。在描述算法的具体实现前，假定本章的研究前提如下：

（1）任务集中所有的任务相互之间独立。

（2）所有任务的截止时限是给定的，不能改变。

（3）任务的开销，如到达时间、任务周期和执行时间是给定的，不允许改变。

（4）如果所有任务的截止时限都满足，那么调度算法是可行；如果有一个任务错过了截止时限，调度算法是失败的。

（5）任务的到达时间是独立的，即一个任务的运行不影响其他任务的到达时间。

（6）任务的执行时间是固定值，不同的任务可能有不同的执行时间；一个任务的执行可能会影响其他任务的完成时间，但是不会影响其他任务的运行时间。

（7）任务抢占的开销可以忽略不计。

（8）任务在不同的核上迁移开销忽略不计（任务在其他处理器恢复运行）。

（9）软件和硬件没有错误。

（10）处理器的速度不变。

（11）一个任务不能同时在两个处理器核上执行，一个处理器核不能同时运行两个或两个以上的任务。

任务调度模型可以表述如下：周期任务集中的每个任务$T_i = (e_i, p_i, d_i)$有三个给定的系数，$1 \le i \le n$，其中，e_i表示任务的最差执行时间，p_i表示任务的执行周期，最差执行时间不能超过周期，则有$e_i \le p_i$。任务T_i中的作业$J_{i,j}$周期性释放，其中 i 表示该作业为任务T_i的作业，j 表示该作业是任务T_i的第 j 个作业，且上一个作业运行的延迟不会影响下一个作业的释放。算法研究基于 *implicit deadlines* 任务集，即所有任务的截止时限等于他们的执行周期，即$d_i = p_i$。因此每个

任务T_i可以表示为$T_i = (e_i, p_i)$。每个任务的利用率为$u_i = (e_i/p_i)$，在任务集中，任务利用率最高的任务其利用率值为 α，即$\alpha = \max\{u_i, i = 1,2,\ldots,n\}$，所以，任务集中的所有任务均满足$0 < u_i \leq \alpha \leq 1(i = 1,2,\ldots,n)$。任务集的总利用率为$U = \sum_{i=1}^{n} = (e_i/p_i)$。

3.3 基于簇集的多核混合调度算法

为了更好的描述基于簇集的混合 EDF 调度算法，本节首先简要介绍了全局 EDF-US[1/2]调度算法，随后，阐述基于簇集的混合 EDF 调度算法的基本思想，并给出了算法的伪代码。

3.3.1 全局 EDF 调度算法

EDF 调度算法中，作业的优先级根据任务的截止时限来确定，假定当前时间为 t，d_i表示任务的相对截止时限，作业的绝对截止时限则为$r_i + d_i$，任务的绝对截止时限越近，任务的优先级越高，任务的绝对时限越远，任务的优先级越低。若作业$J_{i,k}$和作业$J_{j,k}$的相对截止时限分别为d_i和d_j，若$r_i + d_i < r_j + d_j$，则作业$J_{i,k}$的优先级高于$J_{j,k}$。

EDF 调度算法是动态优先级调度算法，在单核调度中是最优的，也就是说，当一个任务集不能使用 EDF 调度算法得到可行的调度顺序，那么其他的调度算法也不能做到。在 EDF 调度算法中，对于一组给定的任务集，任务可调度的充分必要条件为公式（3.1）所示：

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{3.1}$$

由此可知，基于动态优先级的 EDF 调度算法的显著优点是其可调度上限为 100%。

将 EDF 调度算法扩展到多核平台上，全局 EDF 调度算法仍然根据绝对截止时限确定优先级，选择优先级最高的 m 个任务到包含 m 个核的多核架构上进行调度。但是由于 Dhall 效应，全局 EDF 调度算法在多核平台中并不是最优的，其利用率不能达到 100%。

EDF 调度算法在多核平台上有以下优点[37]：

（1）EDF 调度算法易于实现；

（2）使用全局 EDF 调度算法调度一组作业集时，作业间抢占的次数最多为作业集中的作业数，系统中的上下文切换次数最多为作业数的两倍；

（3）由优点（2）可知，作业在核间的迁移次数最多为作业的个数。

虽然研究表明 PFair 公平调度算法拥有更好的利用率，其在多核平台上是最优的实时调度算法，但是其抢占和迁移的开销非常大。同时，全局 EDF 调度算法

实现简单，抢占次数和迁移次数都有上限，因此上下文切换的开销得到了有效的降低，使得全局 EDF 调度算法同样得到了广泛的研究和应用。

由第二章的论述可知，很多研究学者对全局 EDF 调度算法进行了研究，并给出了其利用率界限，对于全局 EDF 调度算法，其利用率上界接近于 $m/2$。其中，Baker 等人[19]在 EDF-US[ζ]算法的基础上提出了改进算法 EDF-US[1/2]，该算法利用率界限为$(m+1)/2$。文献指出对于任意周期任务系统 τ，如果满足以下两个属性：

属性 1：$\alpha \leq 1/2$；

属性 2：$\sum_{i=1}^{n} u_i \leq \frac{m+1}{2}$

该系统使用全局 EDF 调度算法是可调度的，即所有任务都能在其要求的截止时限前得到执行。基于以上结论，EDF-US[1/2]调度算法按照以下规则给任务分配优先级：给利用率大于 1/2 的任务赋予最高的优先级，其他的任务按照 EDF 调度算法分配优先级。在包含 m 个核的多核架构中，任务集的利用率满足 $\sum_{i=1}^{n} u_i \leq (m+1)/2$时，使用 EDF-US[1/2]调度算法是可调度的。基于 EDF-US[1/2]调度算法，本文提出混合 EDF 调度算法。

3.3.2 算法描述

上节简要介绍了 EDF-US[1/2]调度算法，现在来讨论基于簇集的 EDF 混合调度算法的主要思想及其具体实现。假定多核处理器中一共包含 m 个处理器核，将整个多核架构平均分成 m' $(1<m'<m)$个簇集，那么每个处理器簇集内的处理器核数为 m/m'个。

改进算法的主要主要思想是：

首先，将任务集中的任务按照利用率从大到小进行排列，并把排序后的任务依次加入到全局调度队列 GL，同时，每个簇集建立本地的调度队列 LL_i；

其次，从全局调度队列 GL 中依次选取任务并将任务分配到各个簇集，任务分配算法基于启发式 WF 算法进行改进。为了使簇集内的全局 EDF 调度算法能够进行调度，每个簇集中利用率超过 1/2 的任务数不能超过簇集中的核数 m/m'。在进行任务分配时，当任务的利用率大于 1/2（将利用率大于 1/2 的任务称为繁重任务）时，首先检查剩余利用率最大的簇集中已分配了多少繁重任务，若已分配的繁重任务小于簇集内的的处理器核数，则将该任务分配给这个簇集；若已分配的繁重任务等于簇集内的的处理器核数，就选取剩余利用率次大的簇集并检查是否可以进行分配直到该任务完成分配，或没有适合的簇集可以容纳该任务，任务分配失败；当任务的利用率小于 1/2 时，总是将该任务分配到剩余利用率最大的簇集中；

最后，任务分配成功后，将任务加入到簇集的本地调度队列 LL_i；各个簇集根据全局 EDF-US[1/2]调度算法调度本地调度队列 LL_i 中的任务，将利用率超过 1/2 的所有任务赋予最高的优先级，其余的任务按照 EDF 调度算法分配优先级。

因此，改进调度算法主要分为两部分：任务分配算法和簇集内的全局调度算法。分配算法的伪代码如算法 3.1 所示。

算法 3.1：分配算法

输入：任务集 $s=\{\tau_1,\tau_2,\ldots,\tau_n\}$，处理器簇集$\{p_1,p_2,\ldots,p_{m'}\}$；

```
1. begin
2. Sort the task sets by descresing order, u1≥u2≥...≥un, and join those tasks to global
   queue GL=τ1τ2...τn;
3. i:=1; j:=1; flagj:=m/m'; // i=i^th task, j=j^th cluster, flagj represents the rest core in
   cluster used to assigning the highest priority.
4. While(i≤n) do
5.     Select a cluster Cj which utilization is biggest;
6.     if(ui≤Uj)
7.         if(ui>1/2)
8.             if(flagj>0)
9.                 Uj:=Uj+ui;
10.                flagj:=flagj-1;
11.                LLi:=τi; //join the task to clster Cj, and join the task to
   local queue LLi;
12.            else
13.                select another available cluster, goto 6,
14.                if ( there is no available cluster left)
15.                    return false; break;
16.                end if;
17.            end if;
18.        else
19.            Uj:=Uj+ui;
20.            LLi:=τi; //join the task to clster Cj, and join the task to local queue LLi;
21.        end if
22.    else
23.        return false; break;
24.    end if;
25. end while
26. return true;
27. end
```

将任务分配到各个簇集后，在簇集内使用全局 EDF-US[1/2]调度算法调度簇

集本地队列中的任务，具体实现如算法 3.2 所示。

算法 3.2：簇集内全局算法

输入：本地调度队列 LL_i;

```
1. for i=1 to m/m'
2.     if u_i>1/2
3.         Allocate the highest priority to task;
4.     else
5.         The task's jobs allocate priorities according to EDF algorithm;
6.     end if;
7. end
```

下面将运用实例说明算法的具体运行。例如，给定一个包含 4 个核的同构多核结构，将其平均分成两个簇集 C_1={$core_1$, $core_2$}和 C_2={$core_3$,$core_4$}。给定一个任务集 $s=\{\tau_1,\tau_2,\ldots,\tau_8\}$，其中 $\tau_1=\tau_2=(2,3)$，$\tau_3=(4,6)$，$\tau_4=\tau_5=(1,3)$，$\tau_6=\tau_7=\tau_8=(2,6)$。将任务按利用率从大到小进行排列，并依次将任务加入全局调度队列 $GL=\tau_1\tau_2\tau_3\tau_4\tau_5\tau_6\tau_7\tau_8$，按照分配算法 3.1 得出簇集 C_1 包含任务 $\{\tau_1,\tau_3,\tau_6,\tau_8\}$，并将其加入本地队列 $LL_1=\tau_1\tau_3\tau_6\tau_8$，簇集 C_2 包含任务 $\{\tau_2,\tau_4,\tau_5,\tau_7\}$，同样的，将任务加入簇集的本地调度队列 $LL_2=\tau_2\tau_4\tau_5\tau_7$。分别调度簇集 C_1 和 C_2 的本地队列，调度情况如图 3.5 所示，在簇集 C_1 中，任务 τ_1 和 τ_3 有最高的优先级，分别在核 1 和核 2 运行，当任务 τ_1 和 τ_3 运行完后，按照 EDF 算法规定的优先级调度簇集中的其他任务，同理，在簇集 C_2 中，任务 τ_2 具有最高的优先级，其他任务按照全局 EDF 调度算法分配优先级。

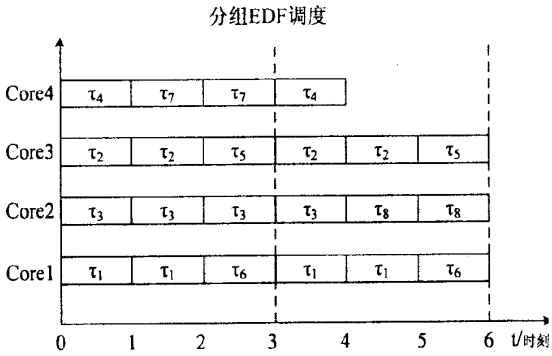

图 3.5 基于簇集的混合 EDF 调度算法调度情况

3.4 利用率分析

为了分析基于簇集的混合调度算法的利用率界限，借鉴局部调度算法界限分析方法。首先假定任务集中所有的任务相互之间独立。在任务集中，任务利用率最高的任务其利用率为 α，即 $\alpha=\max\{u_i,i=1,2,\cdots,n\}$，所以，任务集中的

所有任务均满足$0 < u_i \leq \alpha \leq 1(i = 1 \ldots n)$。任务集的总利用率为$U = \sum_{i=1}^{n}(e_i/p_i)$。进一步的，本文引入了任务集可调度性定义。

定义 3.1：对于一个给定的调度算法，如果一个任务在使用该调度算法调度时，在最差执行情况下的调度时间少于或等于它的截止时限，则称该任务是可调度的。同样的，如果一个任务集中所有的任务根据给定的算法是可调度的，该任务集被认为是可调度的[3]。

对于本文提出的基于簇集的混合 EDF 调度算法，令簇集内的全局调度算法其利用率为s，当簇集内分配的所有任务的利用率总和满足 $U \leq s$ 时，该任务集使用本文提出的算法是可调度的。簇集内的全局调度算法使用 EDF-US[1/2]调度算法，因此，利用率为$s = \frac{m/m'+1}{2}$。令$\beta = \lfloor \frac{s}{\alpha} \rfloor$，取其下限。

定理 3.1：任务数 n 满足$n \leq m'\beta$时，任务集是可调度的。

证明：因为α为任务集中利用率最高的任务其利用率的值，那么其他任务的利用率均小于 α，且 s 为每个处理器簇集可调度的利用率上限，那么每个簇集至少可调度s/α个任务，由于$\beta = \lfloor \frac{s}{\alpha} \rfloor$，所以，对于包含有$m'$个簇集的多核系统，至少可调度$m'\beta$个任务，定理得证。

定理 3.2：当任务集中任务数 n 大于$m'\beta$时，利用率下界为公式（3.2）所示。

$$U_{HEDF} = \frac{(m + m')(m'\beta + 1)}{2(\beta + 1)} \tag{3.2}$$

对于一个给定的任务集，其总利用率为 U，若$U \leq U_{HEDF}$时，该任务集可以使用改进 EDF 混合调度算法进行调度，若$U > U_{HEDF}$时，任务集不能保证能被改进调度算法调度，即可能可以调度，也可能无法调度。为了证明该利用率界限，主要分为两个步骤进行证明：

（1）寻找一个最低界限，当任务集满足$U \leq U_{HEDF}$时，使用基于簇集的混合 EDF 调度算法是可调度的；

（2）当$U > U_{HEDF}$时，至少存在一个任务集，使用基于簇集的混合 EDF 算法是不能调度的。

证明：第（1）部分证明；

假定一个任务集包含有 n 个任务$\{T_1, T_2, \ldots, T_n\}$，该任务集中一部分任务已成功分配到各个处理器簇集，而另一部分任务不能得到分配。假定任务集中顺利分配到各个处理器簇集的任务一共有 t 个，T_k为不能顺利分配的任务中的第一个任务，其任务利用率为u_k。由算法 3.1 可知，任务分配失败的情况为 15 行和 23 行两处。算法中 15 行处产生失败的原因是，任务集中利用率大于 1/2 的任务数超过多核中的核数，因为在分配算法前，任务集中的任务已按照利用率从大到小进行了排序，所以，所有已分配的任务均大于任务T_k的利用率u_k，满足公式（3.3）所示：

$$\underbrace{u_1 \geq u_2 \geq \cdots \geq u_t}_{m} \geq u_k \tag{3.3}$$

因为$u_k > 1/2$，结合公式（3.3）可得公式（3.4）：

$$u_1 \geq u_2 \geq \cdots \geq u_t \geq u_k > \frac{1}{2} \tag{3.4}$$

任意一个簇集的利用率为 s，满足公式（3.5）：

$$s = \frac{\frac{m}{m'} + 1}{2} \tag{3.5}$$

对于每个簇集，假定簇集 C_j 中已分配的任务其利用率总和为 U_j，此时簇集中利用率大于 1/2 的任务数等于簇集中的核数，则有公式（3.6）：

$$U_j = \sum_{i=1}^{m/m'} u_i > \frac{1}{2} \cdot \frac{m}{m'} \tag{3.6}$$

当再次分配一个利用率大于 1/2 的任务时，将超过簇集的利用率 s，如公式（3.7）所示：

$$U_j + u_k > \frac{1}{2} \cdot \frac{m}{m'} + \frac{1}{2} = s \tag{3.7}$$

对上述公式（3.7）进行整理变形，即有公式（3.8）。

$$u_k > s - U_j \tag{3.8}$$

对于算法 3.1 中 23 行处的失败分配，仍然假定 T_k 为余下的未被分配的任务集中的首个待分配任务，U_j 为每个簇集中分配的任务的利用率总和。23 行分配失败的原因是任务 T_k 无法分配到剩余利用率最大的簇集中，可知任务 T_k 的利用率超过了任意一个处理器簇集内可用的剩余利用率，因此，任务 T_k 满足公式（3.8），对公式（3.8）进行恒等变形，则有公式（3.9）：

$$U_j > s - u_k, (j = 1, 2, \ldots, m') \tag{3.9}$$

令任务集的总利用率为 U，则有公式（3.10）：

$$U = \sum_{i=1}^{n} u_i \geq \sum_{i=1}^{k} u_i = \sum_{j=1}^{m'} U_j + u_k \tag{3.10}$$

对公式（3.9）求和可得公式（3.11）：

$$\sum_{j=1}^{m'} U_j > \sum_{j=1}^{m'} (s - u_k) = m'(s - u_k) \tag{3.11}$$

将公式（3.11）代入到公式（3.10），可得公式（3.12）：

$$\sum_{i=1}^{k} u_i > m's - (m' - 1)u_k \tag{3.12}$$

因为任务集中的所有任务在进行分配前都已按照任务利用率从大到小进行排序，所以任务 T_k 的利用率u_k小于所有已被分配的任务的利用率值，则有公式

（3.13）：

$$u_k \leq \frac{\sum_{i=1}^{k} u_i}{k} \tag{3.13}$$

将公式（3.13）代入式（3.12），并对其进行恒等变形，有公式（3.14）：

$$\sum_{i=1}^{k} u_i > \frac{ksm'}{k+m'-1} \tag{3.14}$$

令$f(k)=\frac{ksm'}{k+m'-1}$，对其进行求导可得公式（3.15）：

$$f'(k)=\frac{sm'(m'-1)}{(k+m'-1)^2} \tag{3.15}$$

因为至少会将多核中的多个处理器核分成一个以上的处理器集群，即 $m'>0$，所以$f'(k)>0$，函数$f(k)$单调递增，而 k 的取值范围为$\beta m'<k\leq n$，取最小值$k=\beta m'+1$，并将其代入公式（3.14），结合公式（3.10）可得公式（3.16）：

$$U > \frac{sm'(\beta m'+1)}{\beta+1} \tag{3.16}$$

结合整个分析推理过程，由公式（3.16）可知，对于任意利用率大于$\frac{sm'(\beta m'+1)}{\beta+1}$的任务集，使用算法 3.1 所示分配算法不能完成该任务集的分配，即使用基于集群的混合调度算法是不能调度的。相反的，任务集的利用率小于或等于$\frac{sm'(\beta m'+1)}{\beta+1}$时，使用改进的调度算法，任务集是可调度。第（1）部分得证。

第（2）部分证明：

存在任务集$U=s(\beta m'+1)/(\beta+1)+\varepsilon$，使用算法 3.1 时不能得到分配。证明该任务集不能得到分配的方法分为三步：

一、存在一个这样的任务集

假定存在一个这样的任务集$\tau=\{T_1,T_2,\ldots,T_n\}$，任务集中一部分任务的利用率如下：

$u_i=\frac{\varepsilon}{m'}$，其中$i=1,2,\ldots,n-m'\beta-1$；

另一部分任务的利用率为：

$u_i=\frac{s}{\beta+1}+\frac{\varepsilon}{m'}$，其中$i=n-m'\beta,\ldots,n$；

可知，该任务集包含两种不同利用率的任务，其中(n-$m'\beta$-1)个利用率为 ε/m' 的任务，$m'\beta+1$个利用率为$\frac{s}{\beta+1}+\frac{\varepsilon}{m'}$的任务，该任务的利用率总和为$U=s(\beta m'+1)/(\beta+1)+\varepsilon$。

二、任务集满足假定条件

令 ε 为无穷小，即$\varepsilon\to 0^+$，那么有公式（3.17）：

$$u_i = \frac{\varepsilon}{m'} < \alpha \tag{3.17}$$

另一部分任务则满足下式（3.18）：

$$u_i = \frac{s}{\beta+1} + \frac{\varepsilon}{m'} = \frac{s}{s+\alpha}\alpha + 0^+ \le \alpha \tag{3.18}$$

综合式（3.17）和式（3.18）可得，该任务集中的所以任务都满足假定条件。

三、任务集不能被分配

首先对利用率高的任务进行分配，即首先分配利用率为$\frac{s}{\beta+1}+\frac{\epsilon}{m'}$的$m'\beta+1$个任务，那么，每个处理器簇集将被分配至少$\beta+1$个任务，这是不能被分配的，违反定理 3.1，由此可知，总利用率为$s(m'\beta+1)/(\beta+1)+\varepsilon$的任务集不可调度。

由以上证明可知，基于簇集的混合 EDF 调度算法，当其任务数n满足$n>\beta m'$时，其利用率界限为$U=\frac{s(\beta m'+1)}{\beta+1}$。

因为$s=\frac{m/m'+1}{2}$，将其代入$U=\frac{s(\beta m'+1}{\beta+1}$，可得公式（3.19）：

$$U = \frac{(m+m')(m'\beta+1)}{2(\beta+1)} \tag{3.19}$$

由以上分析可知，混合 EDF 调度算法的利用率为公式（3.20）：

$$U_{HEDF} = \frac{(m+m')(m'\beta+1)}{2(\beta+1)} \tag{3.20}$$

其中m为处理器核数，m'为处理器簇集数。

3.5 算法比较分析

在得出混合 EDF 调度算法的利用率下界后，为了评估混合调度算法的性能，将混合调度算法与全局调度算法和局部调度算法进行比较分析。

3.5.1 混合调度算法与全局调度算法的分析比较

全局 EDF-US[1/2]调度算法的利用率为公式（3.21）所示：

$$U_{EDF-US[1/2]} = \frac{m+1}{2} \tag{3.21}$$

从上述分析可知，混合调度算法的利用率下界为公式（3.20）所示，因为混合 EDF 调度算法至少将处理器分成一个以上的簇集，因此$m'>1$，且$0<u_i\le\alpha\le1$，$\beta=\lfloor\frac{s}{\alpha}\rfloor$，所以$\beta\ge1$，可得公式（3.22）：

$$m'\beta > \beta \tag{3.22}$$

由公式（3.22）可得公式（3.23）：

$$m'(m\beta+m'\beta+1) > (m\beta+\beta+1) \tag{3.23}$$

所以根据公式（3.23）可得式（3.24）：

$$(mm'\beta + m + m'^2\beta + m') > (m\beta + m + \beta + 1) \tag{3.24}$$

对上式（3.24）进行变形，得式（3.25）：

$$(m + m')(m'\beta + 1) > (m + 1)(\beta + 1) \tag{3.25}$$

由式（3.25）可知公式（3.26）成立。

$$\frac{(m + m')(m'\beta + 1)}{(2\beta + 1)} > \frac{m + 1}{2} \tag{3.26}$$

针对上述证明可知，对于有m个相同处理能力的多核处理器，使用混合 EDF 调度算法比单纯使用全局 EDF 调度算法拥有更高的可调度性。

3.5.2 混合调度算法与局部调度算法的分析比较

从可调度性分析中可知，对于局部调度算法，每个核作为一个独立的调度单位，任务不允许在核间迁移，其可调度的界限为公式（3.27）所示[45]

$$U = n - (n - 1)\alpha \tag{3.27}$$

公式（3.27）中 n 为处理器核数，α为任务集中利用率最高的任务的利用率值。因为任务集中所有任务利用率都满足$0 \le u_i \le \alpha < 1$，所以令公式（3.12）中的$u_k = \alpha$，混合调度的界限可为公式（3.28）：

$$U' = m's - (m' - 1)\alpha \tag{3.28}$$

为了比较局部调度和混合调度的可调度性，令处理器核数为 m，则局部调度的调度界限为$U = m - (m - 1)\alpha$，混合调度中将 m 个处理器分成m'组，则公式（3.28）中的$s = \frac{m/m'+1}{2}$，将其代入公式（3.28）可得公式（3.29）：

$$U' = \frac{m + m'}{2} + (m' - 1)\alpha \tag{3.29}$$

为了比较局部调度算法和混合调度的性能，将局部调度算法的调度界限与混合调度的调度界限相减，可得公式（3.30），公式如下所示：

$$U_p - U_h = (m - m')(\frac{1}{2} - \alpha) \tag{3.30}$$

对于混合调度算法，每个簇集中包含有两个或两个以上的处理器核，所以$m' < m$，当$\alpha > 1/2$，任务集中存在有利用率大于 1/2 的任务时，混合调度算法的利用率高于局部调度算法，反之，当$\alpha < 1/2$时，任务集中所有任务的利用率均不大于 1/2 时，局部调度算法对处理器的利用率更高。局部调度算法不允许任务迁移，任务一旦分配后，只能在预先分配的核上执行。因此，当任务集中存在利用率很高的任务时，使用局部调度算法将利用率很高的任务分配到一个处理器核后，导致该处理器核所剩余的利用率很小，无法容纳其他剩余的待分配任务，造成该处理器核所剩余的处理能力永远得不到利用，降低了处理器核的可用处理能力剩余，从而降低了整个多核架构的处理能力。由此可知，在这种情况下，混合调度的性能更好。

3.6 小结

本章首先简要介绍了采用分簇的混合调度算法的调度框架，并通过实例分析了分簇混合调度算法的优点。在全局 EDF-US[1/2]调度算法的基础上，提出了一种基于簇集的混合 EDF 调度算法，并描述了算法的伪代码，对算法的执行过程进行了分析。在此算法的基础上，给出了基于簇集的混合调度算法的可调度性分析，同时，求得了改进算法的利用率下界。最后，对该混合调度算法和全局 EDF-US[1/2]调度算法及与局部调度算法进行比较分析，混合调度算法有更好的可调度性。

第4章 面向共享缓存改进的多核实时调度算法

高速缓存的使用会大大减少处理器存取数据的时间，有效的减少访问延迟，因此，在调度中有效的利用高速缓存资源，可加快任务的运行，提高调度的效率。为了提高多核平台上共享缓存的命中率，减少抖动，本章在混合调度的基础上提出了一种基于共享缓存策略的改进调度算法，该算法将读取相同内存资源的任务尽最大可能的分配到同一个多核处理器簇集，并在各个簇集中使用启发式全局EDF 调度算法，使这些任务尽量同时或前后调度。研究表明算法在保证延迟上界的前提下，相比于不采用共享缓存策略的混合调度算法，共享二级缓存的命中率平均提高 6.86%，从而可加快调度的效率。

4.1 高速缓存对多核调度的影响

近十几年来，处理器的速度越来越快，一个微处理器核从内存中读写一个值的时间可以完成上百次操作，瓶颈存在于存储速度[46]，因此在处理器与内存间引入了多级 Cache 结构，以缓解处理器速度与内存速度的不匹配。随着多核处理器的普及，这个问题更加恶化。在多核架构中，多采用多级 Cache 结构，但是不同于单核处理器，多核处理器的 Cache 结构组织方式更多种多样。如下图 4.1 所示，每个核拥有私有的一级 Cache 或私有的一级 Cache 和二级 Cache，共享二级 Cache 或共享二级 Cache 和三级 Cache。

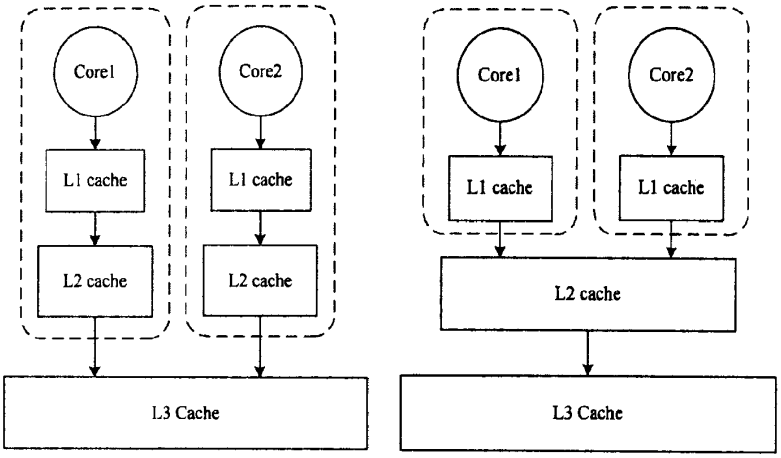

图 4.1 多核架构中多级 Cache 结构示意图

在多核调度中，Cache 对调度的影响主要表现在：

（1）对于私有的 Cache 结构，在全局调度算法中，一个任务从一个核迁移到另一个核执行，需要额外的通信开销，并且加剧了 Cache 的缺失率。例如，任务 τ 先在核 1 上执行，由于某种原因迁移到核 2 执行时，核 2 的私有 Cache 中几乎

没有它所需的信息，需要到原来的核 1 的私有 Cache 中去读取相关资源，这需要开销，对于全局调度而言，频繁的迁移会使这个开销变得很大，尤其当处理的任务数量庞大时。

（2）对于共享的 Cache 结构，多核架构中各个核竞争使用单一的共享 Cache。对于同时运行的任务或线程，因为共享缓存的分配由硬件控制，可能导致不公平的缓存空间大小分配，而不公平的缓存分配将影响线程的运行。对于需要经常读取数据资源的任务，任务的运行不仅依赖于 CPU 所分配的时间片，还依赖于共同调度的线程所占有的共享高速缓存的空间大小。如图 4.2 所示，线程 1 和线程 2 共同调度，线程 1 获得了更多的共享高速缓存空间，导致线程 2 所需高速缓存空间不够，使得执行线程 2 所需的数据频繁的被替换，这将延迟线程 2 的运行。若线程 2 的优先级高于线程 1，对于线程 2，因为获得了一个差的调度环境，线程 2 可能比优先级低的线程 1 晚执行完。

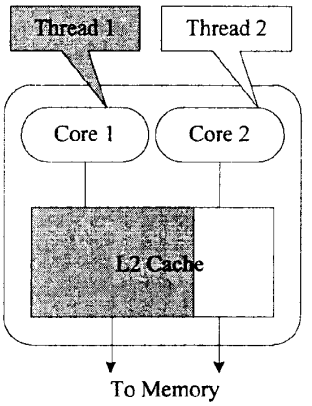

图 4.2 共享缓存对调度的影响

现有的 Cache 替换算法多使用最近最少使用页面置换（LRU，Least Recently Used）算法。传统的 LRU 替换算法把并行执行的多个应用的存储访问平等对待。当多核处理器运行多道程序类应用时，一个应用程序的 Cache 访问失效可能导致另一个应用将要使用到的活跃数据被替换出去，导致系统整体性能的降低和系统执行时间的不可确定[47]。

Fedorova[48]研究了基于吞吐量的系统，它注意到二级缓存不命中的影响远大于一级缓存不命中的影响，这是因为二级缓存不命中可能需要消耗 100-300 个时钟周期去内存读取数据，而一级缓存不命中可以通过二级缓存弥补，只需 10-30 个时钟周期，所以在文献中提出了抑制产生重大的二级缓存冲突的线程共同调度。Chandra 等人[49]的研究指出，相比于低相关冲突的线程，二级缓存上高度冲突的线程共同运行时，线程的执行时间可能上升 65%，如此急剧性能下降是因为急剧增加的二级缓存不命中率。上述研究表明，缓存冲突将造成 Cache 命中率下降，Cache 失效需要一定的时间，将对任务调度的性能带来影响，而相对于一级缓存，

二级缓存的失效影响更大。共享缓存成为系统瓶颈，提高共享缓存的命中率的主要方法有：第一，增大共享缓存的大小，缓存空间的增大可以有效提高缓存命中率，但是，硬件实现成本很高；第二，通过改进算法来提高共享缓存的命中率，该方法主要通过软件技术实现。

4.2 相关的改进方法

由上一小节的介绍可知，在多核上同时运行的线程或任务，他们共享一个单一的二级缓存。一个线程占有高速缓存的大小将影响它的高速缓存命中情况，从而影响线程的执行。因此，这就要求我们在多核上进行任务调度时考虑共享二级缓存对调度带来的影响。针对以上问题，采用软件进行改进的主要解决方法分为两种：一是在调度过程中选择相应的线程共同调度，二是通过软件的方法对共享的二级缓存进行划分。

选择相应的线程进行共同调度的方法在单核多线程调度中曾有过研究，既然运行的多个线程同时进行调度会产生冲突，那么就选择产生冲突最小或不会产生冲突的线程共同调度。Kihm 指出通过获得每个线程的 Cache 访问信息，然后根据这些统计信息可以获得线程未来对 Cache 的访问情况。操作系统根据预测的结果选择可能产生冲突最小的线程共同调度[47]。

为了有效的解决共享缓存造成的任务运行过程中优先级反转的问题，Fedorova[50]提出了一个改进的调度算法。通过分析每个线程平均使用共享二级缓存时线程的性能表现，算法重新分配各线程获得的 CPU 运行时间。 如果一个线程因为不公平的共享高速缓存分配，导致线程的性能表现下降，那么该线程获得更多的 CPU 执行时间，反之亦然。算法的难点在于如何确定线程的表现下降是因为不公平的共享高速缓存分配，文献指出通过分析执行时间统计模型可获得，而不需要增加新的硬件结构。

Anderson 等人指出当共同调度的线程所需的内存资源大于共享的二级缓存的容量时，线程运行时必将产生共享缓存冲突，并且其中一个或多个线程所需资源必须要频繁的换进换出才能解决这种冲突。针对这一问题，Anderson 等人提出在算法中引入相关策略抑制二级缓存冲突的线程共同调度。并提出了多核平台上针对软实时系统的基于缓存感知的按比例公平调度算法（PFair）[25]，为了抑制相关任务共同调度，提高共享缓存的命中率，算法中首次提出了 megatask 任务的概念，一个 megatask 任务中包含多个任务，但其被当作一个统一的调度实体，这就限定了 megatask 中的任务不能同时调度，从而使共同调度的线程所需内存资源不大于共享缓存的空间大小。因此，算法在进行调度前先将会产生二级缓存冲突的任务分成组，分别为 megatask 和自由任务，算法采用分层次的调度方法，根层调度采用 PD^2 调度算法调度所有的 megatask 和自由任务，再用 PD^2 算法调度

megatask 里面的任务，为了不使 megatask 任务组里的任务错过截止时限，文章通过轻微调整 megatask 任务中各任务的权值来有效地避免。

来自同一研究小组的研究[26]指出鼓励来自同一应用的多线程（MTT，共享同一的工作区）共同调度可以降低线程间的影响。通过提前释放子任务或作业使得相同 MTT 中的多个线程执行时间不间隔很远。文章分别提出了关于 PD^2 和全局最短截止时间（G-EDF）调度算法的改进方法 。

在保证实时性的前提下抑制有高速缓存影响的任务共同调度，或鼓励读取相同资源的任务共同调度，都可以改进高速缓存性能。文献[27]通过同时应用鼓励共同调度和抑制共同调度这两种策略，提出了一种基于全局 EDF 调度算法的启发式算法，该算法通过跟踪高速缓存的使用情况，使用不同的调度策略，改善多核平台上共享高速缓存的表现。文献假定同一多线程任务集（MTT）中的线程读取相同的缓存空间（WSS）， 因此，算法鼓励同一 MTT 中的线程共同调度。当共享二级缓存的利用率不高时，提升需要缓存空间（WSS）最大的 MTT 的优先级；当共享二级缓存利用率很高时，则提升所需空间（WSS）少的任务进行调度。

文献[51]通过研究在多核结构中共享高速缓存对线程最差执行时间的影响，提出了一种贪婪的多核调度算法。算法将最差执行时间的分析结果融入到多核调度中，通过共同调度核间影响最小的线程以降低高速缓存的不命中率，该算法不针对具体的算法，既适用于局部调度又适应于全局调度。

另一种方法是不改变共同调度的线程，通过软件的方法划分 Cache 空间以减小共享高速缓存对调度的影响。

Anderson 等人提出了鼓励或抑制共同调度任务来改进高速缓存的性能。但是所有这些工作都是在假定实时线程的最差执行时间（WCET）是已知的这一基础上的，在他们建立的系统模型中仍然不知道如何获得每个实时线程的最差执行时间（WCET）。不同于 Anderson 的工作，文献[52]将共享的高速缓存空间进行划分，以避免同时运行在不同核上的任务产生高速缓存干扰。因此能够采用现有的使用在单核上技术分析任务的最差执行时间 WCET，并对系统进行可调度性分析。算法采用 page-coloring 进行高速缓存的划分，并在任务满足以下条件时调度任务：一、作业是所有等待调度的作业中优先级最高的；二、最少有一个空闲的核；三、有足够的划分的高速缓存。

针对以上两种方法，采用鼓励或抑制共同调度的多核调度算法其优点在于最大限度的开发了共享资源，缩短了任务的执行时间，难点是如何确定哪些任务可以共同调度；而对于划分共享 Cache 的调度方法，优点是有效地消除了在不同核上同时运行的任务产生的高速缓存干扰。缺点是将共享的高速缓存私有化，没有达到共享资源的最大化利用。

4.3 系统模型

为了更好的描述改进算法，在介绍算法的具体实现前，本小节将对任务模型、处理器模型以及调度框架进行简要介绍。

4.3.1 任务模型

之前提出的周期任务集模型中，假定任务之间是相互独立的，但在实际应用中，任务之间可能存在联系。

定义 4.1：如果在任务的运行过程中任务读取相同的数据资源，则称该任务之间存在数据依赖。

对于在多核中共同调度的任务，如果它们之间存在数据依赖，那么它们共同调度时将可以提高共享二级缓存的命中率。例如，任务 T_1 和任务 T_2 之间存在数据依赖，如果共同调度任务 T_1 和任务 T_2，即任务 T_1 在核 1 上运行，任务 T_2 同时在核 2 上运行，那么当任务 T_1 首次将共同读取的资源调入共享的二级缓存后，任务 T_2 之后要读取该资源时，将不需要去内存读取该资源，而是在共享的二级缓存中就能命中该资源，这可以节省大量的时间，加快任务的运行。若将有数据依赖的任务分配到不同的簇集，因为 Cache 块一致性问题，这将会会加重任务调度的开销，同时，若二级缓存上没命中就需去更高级缓存或内存中读取，都将延缓任务的运行。

对于一个确定的应用任务的集合，可以提前预知哪些任务之间存在数据依赖。为了更好的描述任务之间的数据依赖，使任务集模型更符合实际情况。本章在周期任务模型的基础上加入了相关约束，引入符号 r_i（dep_i, num_i），以表示哪些任务之间存在数据依赖，以及数据依赖的程度。所谓数据依赖的程度，即任务集中读取共同资源的任务数与任务集中的任务数的比值。当两个任务的 dep（$dep \neq 0$）值相同时，表示两个任务存在数据依赖，num 值为任务之间数据依赖的程度，当 dep=0 时，代表该任务为独立任务，不与其他任务存在数据依赖。提出符号 r_i 主要用来确定哪些任务可分配到同一个处理器簇集共同调度，实际应用中存在任务之间读取相同内存资源，例如，一个 MPEG 视频帧内有不同的任务处理不同的功能，但是他们读取的内存资源是相同。

在加入表示任务间数据依赖的符号表示后，每一个任务可表示为 $T_i = (e_i, p_i, d_i, r_i)(1 < i < n)$，其中 p_i 表示任务周期，e_i 为任务执行时间，d_i 表示任务的相对截止时限，且任务执行时间小于任务的周期 $e_i \leq p_i$。本文仅考虑任务周期与相对截止时限相等的情况，即 $p_i = d_i$。任务 T_i 的利用率 $u_i = e_i / p_i$，任务集的总利用率 $U_i = \sum_i^n \frac{e_i}{p_i}$。假定任务错过截止时限不会延迟其后续任务的释放。因此，任务集中的任务可以简要表述为 $T_i = (e_i, p_i, r_i)$。

4.3.2 处理器模型

目前，大多数多核处理器采用私有的一级 cache，而二级 cache 或更高层次的 cache 结构的组织方式和管理形式则有所不同，二级 cache 或更高层次的 cache 可能是共享的也可能是私有的。对于私有的二级 Cache，每个核分配同样大小的 cache 空间，cache 不命中时处理器通过内部总线访问其他处理器二级 cache 中是否有需要的资源，若还不命中则访问内存。私有二级缓存的缺点是：二级 cache 的使用开销不平衡，例如，有的核的二级 cache 利用率高，有的核的二级 cache 利用率低；共享的二级 cache 则不存在这个问题，也更利于数据资源的共享；所以，二级缓存多采用共享方式。二级 cache 或三级 cache 共享的方式也不近相同，在部分多核结构中二级 cache 被所有的核共享；有的采用部分核共享二级 cache，所有的核采用三级 cache。

本文考虑在软实时的背景下研究如图 4.3 所示的典型同构多核模型，可知，该模型采用分层次的多级内存结构，一组核共享 L2 缓存，所有的核共享 L3 缓存。其中每个核中任务运行是单线程的。

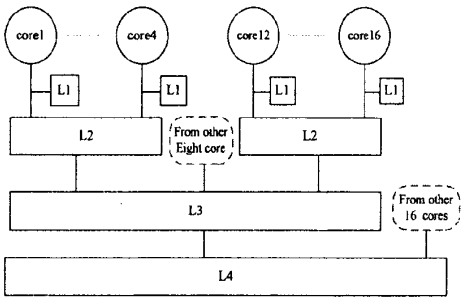

图 4.3 高速缓存层次结构的多核

假定多处理器总共有m个处理器核，令共享二级缓存的处理器核为一组，并作为一个簇集（cluster）。整个多核架构分成m'个簇集，那么每个处理器簇集内的处理器核数为m/m'。

4.3.3 算法步骤

改进算法的过程主要分为两步：任务分配和簇集内的全局调度。在进行任务分配前，需要进行任务管理以需要获得任务参数的相关信息，任务的执行步骤如图 4.4 所示。任务管理主要获得任务的周期p_i，任务最差执行时间e_i和任务之间数据依赖关系r_i。对任务进行整理后对任务进行分配，任务分配过程中将有任务依赖的任务尽量分配到同一个簇集。在簇集内采用改进的全局调度算法，算法鼓励读取相同内存资源的任务共同调度，若任务T_1正在运行，T_2与T_1读取相同内存资源，那么就一直提升T_2的优先级，直到T_2被执行。簇集内全局调度在全局 EDF 调度算法的基础上进行改进，在全局调度中，全局 EDF 调度不是最优的，任务可能会错过它们的截至期限，但是，全局 EDF 调度算法的截至期限的延迟是有限的[53]，

这符合软实时应用的需求。

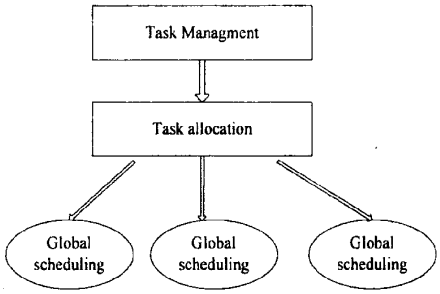

图 4.4 基于共享缓存策略的混合调度算法的步骤

4.4 基于共享缓存策略的混合调度算法

文献[44]在如图 4.3 所示多核架构中提出了混合调度方法，算法将共享缓存的多个处理器核作为一个处理器簇集，将任务静态的分配到各个处理器核簇集，然后在簇集内部采用全局调度算法。但算法没有充分考虑处理器簇集中共享缓存的冲突访问情况。本文在此基础上，设计将读取相同内存资源的任务静态分配到同一个处理器簇集，并在处理器簇集内改进全局 EDF 调度算法，使有数据依赖的任务共同调度，以改进高速缓存的性能，缩短实时任务的实际运行时间，从而可改进任务调度的性能。

4.4.1 任务分配

任务分配阶段的主要任务是将任务集静态的分配到各个处理器簇集。在多种启发式分配算法中，WF 算法和 NF 算法将任务均衡的分配到各个核，使得分配较均衡；而 FF 算法和 BF 算法是使每个核尽量分配更多的任务，从而可能使得其他的处理器核空转，造成整个系统的不平衡。考虑到处理器核的负载基本平衡，本文在 WFD（Worst-Fit Descreasing）算法的基础上提出启发式算法，WFD 算法将待分配任务按处理器利用率递减次序排列，然后按照这个次序采用 WF 算法分配任务到各个处理器核。

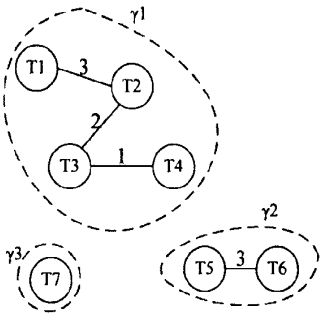

图 4.5 任务合并实例

将读取相同内存资源的任务尽量分配到同一个处理器簇集，可有效提高二级

缓存的命中率，优化任务的运行；再则，读取相同内存资源的任务分配到不同的处理器簇集时，将使各高速缓存中同时保存内存资源的副本，这就造成高速缓存的资源消耗，同时可能引起高速缓存副本不一致的问题。因此，改进算法的主要思想是尽量将有数据依赖的任务分配到同一个处理器簇集，在任务分配前，首先将有任务依赖的任务合并成一个大任务，即若任务 $T_i(e_i,p_i,r_i)$与 $T_j(e_j,p_j,r_j)$中 *dep* 值相等，即 $dep_i == dep_j$，则将这两个任务合并为大任务。若大任务的总利用率超过了簇集中的可利用率u_c，则将大任务中 *num* 值最小的任务独立出来，即令依赖最小的任务为独立任务，直到大任务的总利用率不超过u_c。不与其他任务读取相同

算法 4.1： 分配算法

输入：任务集$\{T_i(e_i,p_i,r_i(dep_i,num_i))\}(0<i\leq n)$，处理器簇集 C_i 的核数 m'；

```
begin
Sort the task set by descreasing r_i_num_i;
for i:=1 to numTasks do //merge task
    for j:=2 to numTasks do
        if ((c_j_dep_j)==c_i_dep_i))&&(u_i<utilization of cluster))
            merge task;
            u_i=u_i+u_j;
            Delete(T_j); //删除任务 T_j
        end if;
    end for;
    enQueue(T_i); //将合并后的任务加入队列；
end for;
Sort the tasks by descreasing order in queue;
While(queue is not null)
    T_i:=dequeue();
    Find the available cluster C_j which utilization is biggest;
    if(u_i<U_j) //U_j:=utilization of C_j;
        Task T_i is allocated to cluster C_j;
        U_j=U_j+u_i;
    else
        return false;
    end if;
end while;
end
```

资源的任务作为独立任务。如图 4.5 所示，有一个任务集包含 7 个任务，其中，

任务T_1、T_2、T_3、T_4有任务依赖，任务T_5、T_6有任务依赖。则将任务T_1、T_2、T_3、T_4视为一个大任务γ_1，任务T_5、T_6视为一个大任务γ_2，任务T_7作为独立任务γ_3。对于大任务γ_1，若其利用率大于处理器簇集的利用率，则选取相关依赖最小的任务T_4作为独立任务。启发式分配算法的具体实现如算法 4.1 所示：

4.4.2 簇集内全局调度

因为高速缓存中的页面替换算法大多使用 LRU 算法，对于共享的二级缓存，刚刚调出缓存的数据可能是其他核上的任务需要读取的，而去内存读取需等待多个时钟周期，所以本文在全局 EDF 调度的基础上，通过启发式的方法提升有数据依赖的任务之间的优先级，使有数据依赖的任务先后执行或同时执行，以提高二级缓存的命中率，从而提高调度的效率。为了更好的解释算法，提出几个相关概念：

算法 4.2： 簇集内的全局调度算法

输入：任务集$\{T_i\}(0<i\leq n)$，处理器簇集 C_i 的核数 m'；

```
for i:=1 to runtime do
Assign each task a priority equal to its deadlines;
enPreQueue(Preemptive task); //Preemptive task join the preemptive queue;
enUrQueue(Urgent task);// Urgent task join the urgent queue;
if(eligible preemptive tasks) then
        Preemptive task running
else
        Continue to run
if(no tasks running)
        if(eligible urgent tasks)
                urgent tasks running;
        else
                tasks in ready queue running
end
```

紧急任务：当一个任务 T_i 被调度了，就绪队列中任务 T_j 的 dep_j 与任务 T_i 的 dep_i 值相等，则将 T_j 被标为紧急任务，并且将其加入紧急队列，当有多个紧急任务时，紧急队列按照全局 EDF 算法设置的优先级对任务进行排列。紧急任务这一策略主要用来鼓励读取相同内存资源的任务共同调度。

抢占任务：指那些当前时间与截止时间的间隔等于任务所需执行时间的任务，已知任务 $T_i(e_i,d_i)$，当前时间为 t，若 $d_i-e_i\leq t+\varepsilon$（$\varepsilon$ 为无穷小），则该任务 T_i 是紧急任务，提出紧急任务是为了更好的保证任务的实时性，虽然全局 EDF 调度算

法截止期限延迟是有界的，但总是提升与当前运行的任务读取相同内存资源的任务，可能导致额外的任务错过其截至期限，所以提出抢占任务这个概念。

算法改进的主要思想：首先根据每个任务的截止时限分配优先级，即按全局 EDF 算法分配任务优先级队列；当出现抢占任务时，任务抢占执行；当前正在运行的任务周期执行完毕时，就调度紧急任务进行调度；若没有紧急任务，则按照全局 EDF 算法调度就绪队列中的下一个任务。算法的具体步骤如算法 4.2 所示。

在没有强占任务时，任务不允许抢占执行，这样可以减少任务间的频繁切换，降低任务切换的开销，并且可以提高任务运行时缓存的命中率。

为了更好的分析簇集内的改进全局调度算法，分析如表 4.1 所示的一系列周期任务使用全局 EDF 调度算法和改进的全局 EDF 调度算法在多核架构簇集上的调度情况，多核处理器簇集内包含有两个相同能力的处理器核。

表 4.1 待任务调度集实例

任务名称	到达时间（时刻）	执行时间（时刻）	周期（时刻）
T_1	0	1	2
T_2	0	1	4
T_3	0	1	4
T_4	0	2	8
T_5	0	2	8

该任务集包含 5 个任务分别为$T_1(1,2)$，$T_2(1,4)$，$T_3(1,4)$，$T_4(2,8)$，$T_5(2,8)$，均在 0 时刻被释放，每个任务的截止时限等于其周期，其中任务T_1和T_3有数据依赖，任务T_2和T_4有数据依赖。对于周期任务而言，查看任务集的运行情况，求解出该任务集中所有任务的公周期，分析该公周期内任务的运行情况即可。

当采用全局 EDF 调度算法调度该任务集时，其调度情况如图 4.6 所示，图中横坐标表示时间，纵坐标表示任务。算法根据任务的绝对截止时限分配优先级，如在 0 时刻，所有任务被释放，任务T_1截止时限为 2，优先级最高，任务T_1获得调度。任务T_2和T_3具有相同的截止时限，但当前只剩一个核可用，按照 FIFO 调度原则，T_2先调度。

采用改进的全局 EDF 调度算法调度该任务集时，算法首先按照绝对截止时限分配优先级，即按照全局 EDF 调度算法的方法初始分配优先级，在运行的过程中，根据任务之间是否有数据依赖提升任务的优先级，上述任务集采用该算法调度时，其调度情况如图 4.7 所示，在 0 时刻，所有任务释放，根据截止时限分配任务的优先级，T_1具有最高的优先级，所以任务T_1首先运行，集合中与T_1有数据依赖的任务T_3被提升为紧急任务，虽然T_2与T_3的截止时限相同，但T_3优先级高于T_2，所以选择T_3运行，在时刻 1，任务T_1和任务T_3的一个作业运行完毕，算法从就绪队列中选取优先级最高的两个作业进行调度，首先，查看是否有抢占任务，没有则查看是

否有紧急任务，都没有则按照全局 EDF 调度算法进行调度，其中T_2的截止时限最近，所以选择任务T_2到核 1 运行，任务T_4与任务T_2有数据依赖，提升其优先级，选择进行调度。

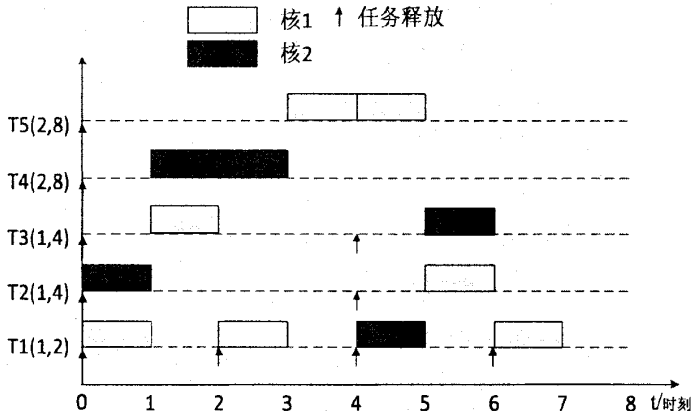

图 4.6 全局 EDF 调度情况

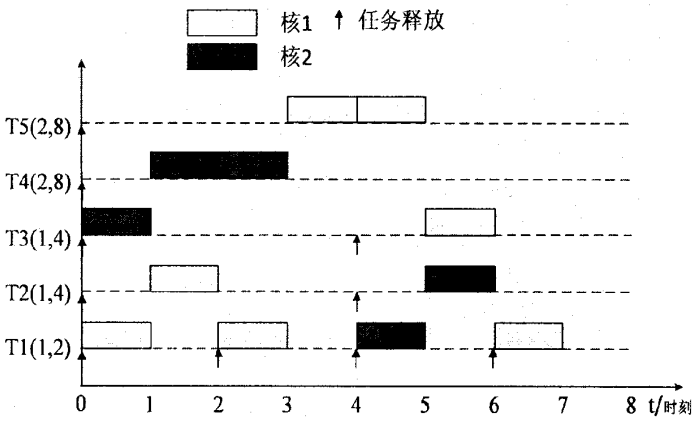

图 4.7 改进 GEDF 调度情况

4.4.3 算法延迟界限分析

面向共享缓存改进的混合调度算法一直提升有数据依赖的任务的优先级，为了分析该算法的是否会造成任务无限的延迟，下面将给出相关的分析。

定义 4.2[54]：（优先级函数）对于每一个作业$T_{i,j}$，给定一个时间函数$\chi(T_{i,j},t)$，其中$t \geq 0$，称该函数为作业的优先级函数，如果满足如下公式（4.1）所示：

$$\chi(T_{i,j},t) < \chi(T_{h,k},t) \tag{4.1}$$

那么在时刻t，作业$T_{i,j}$的优先级高于作业$T_{h,k}$的优先级。

定义 4.3[54]：对于任务T_i，存在常数φ_i和 ψ_i，使得T_i中的每个作业$T_{i,j}$在时刻 t 都满足公式（4.2）：

$$r_{i,j} - \varphi_i \leq \chi(T_{i,j},t) \leq d_{i,j} + \psi_i \tag{4.2}$$

则称该调度算法的优先级是窗口约束的。

改进的算法中通过启发式的方法提升具有数据依赖的作业的优先级，但是其优先级的提升是具有窗口约束的。对于改进的全局 EDF 调度算法，其优先级函数满足窗口约束条件，且其中$\varphi_i = \psi_i = 0$；因此基于共享缓存策略的混合调度算法可以根据文献[54]中提出的公式计算算法的延迟界限。使用改进调度算法进行调度，任务集中的任务 T 其延迟界限为$x + e(T)$，其中 x 的定义如公式（4.3）所示：

$$x = \frac{E_L + A(k)}{M - U_L} \tag{4.3}$$

其中，E_L为任务集τ中$M-1$个执行时间最长的任务执行时间的总和，U_L为任务集τ中$M-1$个利用率最高的任务利用率的总和，且$A(k)$的定义如公式（4.4）所示：

$$A(k) = (M-1)\cdot\rho - e(T) + \sum_{T^j \in \tau / T^k} \left(\left\lceil \frac{\psi_k + \phi_j}{p(T^j)} \right\rceil + 1 \right) \cdot e(T^j) \tag{4.4}$$

因为改进算法没有提前释放任务或运行任务的截止期延后，所以$\varphi_i = \psi_i = 0$，且满足公式（4.5）。

$$\rho = \max_{T^h \in \tau}(\phi_h) + = \max_{T^h \in \tau}(\psi_h) \tag{4.5}$$

所以 ρ=0；因此有公式（4.6）：

$$A(k) = \sum_{T^j \in \tau / T^k} e(T^j) - e(T) \tag{4.6}$$

将公式（4.6）带入公式（4.3），可得算法的截止时限为公式（4.7）：

$$x = \frac{E_L + \sum_{T^j \in \tau / T^k} e(T^j) - e(T)}{M - U_L} \tag{4.7}$$

由式（4.7）可知，算法的延迟界限是有界的，对于软实时系统而言，一定范围的延迟是可以接受的。

4.5 仿真测试

仿真实验是验证多核实时调度性能的有效手段之一。本文采用仿真测试，模拟任务执行的情况，以观察算法的有效性。

4.5.1 处理器及任务集描述

仿真实验设计采用如图 4.8 所示的多核结构，该多核处理器结构有 8 个处理器核，每个处理器核的处理能力相同，每两个处理器核共享一个 8 路组相联 512k 二级高速缓存，高速缓存采用 LRU 替换策略，令共享二级缓存的两个处理器核作为一个处理器簇集，因此，该多核架构一共分为 4 个处理器核簇集。

仿真实验分为三个部分，第一部分是生成任务集，为了更好的分析算法的通用性，任务集由计算机随机产生；第二部分为任务分配算法，算法将任务集中的任务分配到各个处理器簇集；第三部分为簇集内的全局调度算法的实现。

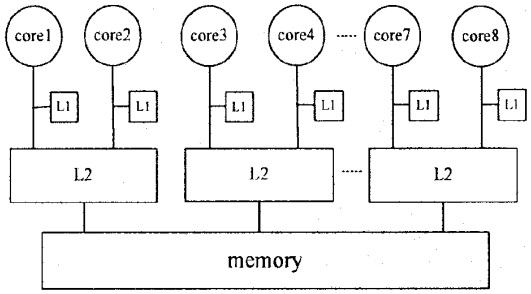

图 4.8 仿真多核结构示意图

生成任务集时按下列方式生成，任务执行周期 D_i 的取值范围为[10，100]，此处，考虑任务的截止时限等于任务周期，每组任务集中任务的个数 n 取值范围为[25，50]，仿真测试的主要目的是考察算法是否有提高任务运行时二级缓存的命中率，所以，尽量使产生的任务集可调度。因此，令任务的利用率 u_i 为服从参数 λ 的指数分布，指数分布的均值为 $1/\lambda$，所以令 λ 的取值范围为$[8/n,9/n]$。则任务的执行时间为 $E_i=u_i*D_i$，随机生成 100 个任务集。假定所有任务均需读取内存资源，且均为 192k 时，任务读取的内存资源由计算机随机生成，取值范围为[1,100]。下表 4.2 描述了任务各个参数的生成情况。

表 4.2 任务各个参数的生成情况

任务参数	生成情况
截止期 D_i	取值范围为[10，100]的随机整数
利用率 u_i	服从参数为 λ 的指数分布，λ 的取值范围为$[8/n,9/n]$
读取资源 c_i	取值范围为[1,100]的随机 3 个整数

第二部分中任务分配算法分别实现了 WFD 分配算法和改进的分配算法，不同的分配算法严格执行同一任务集进行测试比较。

第三部分为簇集内的全局调度，因为算法是在全局 EDF 算法的基础上进行的改进，因此，此部分实现了全局 EDF 调度算法和改进后的基于缓存策略的 EDF 算法。

4.5.2 实验结果及分析

假定 α 为任务集中有任务依赖的任务数占总任务数的比例，为了了解该比例对算法的影响，分别选取不同 α 值的任务集进行测试，α 的取值分别为 α=0.8，0.6，和 0.4。簇集内的全局调度算法采用全局 EDF 调度算法，分别测试使用 WFD 分配算法和改进的分配算法（HWFD）时高速缓存的命中率，测试结果如表 4.3 所示，改进的分配算法二级缓存的命中率有了明显提升，平均提升了 3.76%。

表 4.3 采用不同分配算法时 Cache 失效率测试结果

α	WFD (%)	HWFD(%)	命中率提高(%)
0.8	20.51	16.99	3.52
0.6	22.55	18.71	3.84
0.4	23.18	19.25	3.93
平均	22.08	18.32	3.76

为了测试簇集内的基于缓存策略的 EDF 算法（caEDF）是否有进一步改进二级缓存的命中率，现分配算法统一采用 HWFD 分配算法，在簇集内分别采用全局抢占 EDF 调度算法和基于缓存策略的 EDF 调度算法进行调度，测试结果如表 4.4 所示，由表可知，簇集内采用 caEDF 调度算法时，比采用全局抢占 EDF 调度算法有更好的二级缓存表现，二级缓存的命中率平均提升了 3.10%，同时，相比首先采用 WFD 算法进行分配然后使用全局 EDF 调度的混合调度算法，基于共享缓存策略的混合调度算法的二级缓存命中率平均提高 6.86%。

表 4.4 簇集内采用不同调度算法时 Cache 失效率测试结果

α	全局 EDF(%)	caEDF（%）	命中率提高(%)
0.8	16.99	13.92	3.07
0.6	18.71	15.69	3.02
0.4	19.25	16.05	3.20
平均	18.32	15.22	3.10

为了进一步测试算法，增大共享二级高速缓存的容量，令其大小为 1M。测试 α=0.4 的任务集，簇集内的全局调度算法采用全局 EDF 调度算法，分别测试使用 WFD 分配算法和改进的分配算法（HWFD）时高速缓存的命中率，测试结果如表 4.5 所示，改进的分配算法可以使缓存命中率提高 4.15%。

表 4.5 采用不同分配算法时 Cache 失效率测试结果（cache=1M）

α	WFD (%)	HWFD(%)	命中率提高(%)
0.4	10.40	6.25	4.15

同时，分配算法统一采用 HWFD 分配算法，在簇集内分别采用全局抢占 EDF 调度算法和基于缓存策略的 EDF 调度算法进行调度，共享高速缓存缺失情况如表 4.6 所示，簇集内采用 caEDF 调度算法时二级缓存的命中率平均提升 1.31%。

表 4.6 簇集内采用不同调度算法时 Cache 失效率测试结果（cache=1M）

α	全局 EDF(%)	caEDF（%）	命中率提高(%)
0.4	6.25	4.94	1．31

综合表 4.3、表 4.4、表 4.5 和表 4.4 可得算法的整体比较结果，结果如图 4.9 所示。柱形图的横坐标表示任务集中有任务依赖的任务占任务集中总任务数的比值，即 α 值；纵坐标表示采取不同算法时共享高速缓存的缺失率。其中第二列柱

形整体是 cache 为 1M 时共享缓存的缺失情况，其余列是 cache 为 512*k* 时缓存的缺失情况。由柱形图可清楚的看到，不论 cache 容量的改变，改进的分配算法带来了二级缓存命中率的提升，同时，簇集内采用基于缓存策略的 EDF 调度算法时共享二级缓存的命中率得到了进一步改进。

将读取相同内存资源的任务分配到了同一个处理器簇集，增大了任务所需内存资源在高速缓存的机会。例如，若读取相同资源的任务 T_1 和 T_2 分配到了同一个簇集，任务 T_1 执行时将其所需的资源调入了共享内存，那么当任务 T_2 运行时，所需的资源可能不需要去内存读取，在高速缓存中就可以命中，这就提高了任务运行时缓存的命中率。因此，改进的分配算法能够提高任务执行时二级高速缓存的命中率。从图 4.9 可知，从命中率来说，算法在未做任何改进前，任务集中读取相同内存资源的任务占总任务的比例越高， L2 的失效率越低，这是因为任务集中越多任务读取相同内存资源，随机分配到同一个共享二级缓存核的概率越高，但对本文提出的基于缓存策略的混合调度算法来说，被调度的任务集中读取相同内存资源任务所占的比例对缓存命中率提升影响不大，这可能是由二级缓存的容量大小决定的，即对大小一定的 cache，算法命中率提升空间有限。当我们增大 L2 容量大小，令其为 1M 时，共享高速缓存的命中率得到了显著提升，但增大 L2 容量需要额外的花费。

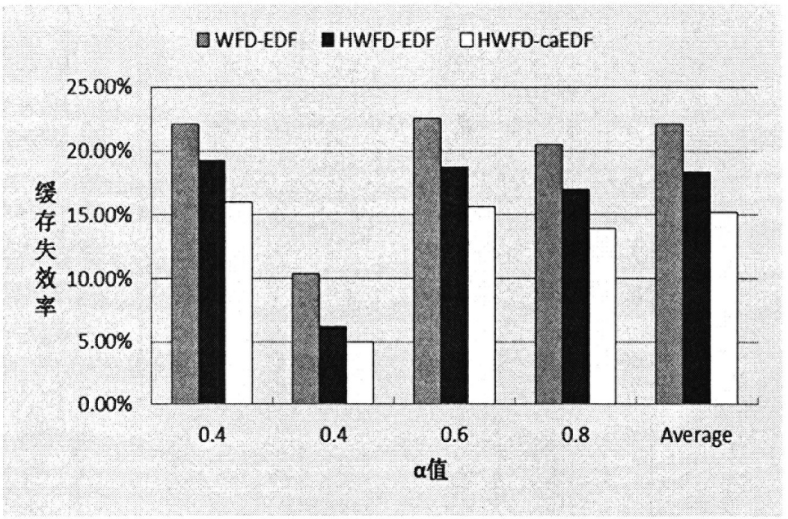

图 4.9 算法失效率比较结果

4.6 小结

在多核实时调度中，不同核上同时运行的任务产生的共享缓存的冲突会影响任务的运行，同时，高速缓存命中率的提升可以有效降低实时任务的执行时间，因此，本文在文献[44]的基础上，充分考虑共享高速缓存对任务调度的影响，提出了一种基于高速缓存策略的多核实时混合调度算法。以共享二级缓存的处理器作为一个簇集，算法首先将读取相同内存资源的任务分配到同一个处理器簇集，

降低簇集内任务运行时共享缓存的失效率，并进一步的在簇集内采用鼓励共同调度的策略。通过模拟实验分析可得，相比于原算法，基于缓存策略的混合 EDF 调度算法有效提高了二级缓存的命中率，从而减少了共享缓存的抖动，加快了任务的运行。

结 论

在传统单核处理器面临瓶颈，传统单核处理器的结构设计必须另辟新的途径时，多核处理器应运而生。随着科学技术的飞速发展，多核处理器被越来越多的运用于实时系统。不同于单核处理器，多核处理器并非多个处理器的简单拼凑，应用于单核处理器的软件不能直接简单的用于多核处理器，需要经过一定的改造。对于实时系统而言，任务调度的重要性不言而喻，单核处理器实时调度已日趋成熟，虽然多核实时调度问题的理论研究开展已久，但是，多核处理器的实现与应用也是近几年才得到广泛应用，多核处理器的任务调度问题还有许多问题需要解决，特别是多核实时调度问题。多核实时调度算法主要分为两类：局部调度和全局调度。局部调度算法不允许任务在不同的核间迁移，降低了多核处理器的并行性；而全局调度算法虽然允许任务在同的核上执行，但是，算法的开销很大，任务上下文切换次数频繁，针对这一问题，学者们提出了结合全局调度和局部调度优点的混合调度算法。同时，在多核结构中多个核共享一个单一的高速缓存，不同核上执行的任务，其高速缓存的空间分配及与其共同执行的任务对高速缓存分配的干扰，都会对调度会产生影响，本文针对这些问题，通过理论分析和实验相结合的研究手段，对课题进行了深入的探讨，并取得了一系列的研究成果。主要工作内容总结如下：

（1）通过对多核实时调度算法现状的分析，对多核实时调度算法的局部调度和全局调度两类方法进行了分类研究，分析并给出了多核局部调度和全局调度各自的优缺点。对混合调度方法进行了深入研究，概述了混合调度的概念及其分类，并分析了了混合调度的研究现状。

（2）提出了一种基于 EDF-US[1/2]调度算法的混合调度算法，对混合调度算法的具体实现进行了描述和分析。在此算法的基础上，对算法的可调度性进行了分析并给出了算法利用率界限。最后，将该算法与全局的 EDF-US[1/2]调度算法和局部调度算法的利用率进行了比较分析，混合调度算法实现了更高的可调度性。

（3）提出了一种面向共享缓存改进的混合调度算法，算法主要适用于软实时系统。在分析了共享高速缓存对多核实时调度影响的基础上，通过将读取相同内存资源，即有数据依赖的任务分配到同一个处理器簇集，并在处理器簇集内使用相关策略提升有数据依赖的任务的优先级。鼓励有数据依赖的任务在簇集内共同调度。此外，对算法的任务模型，调度框架，及其具体实现进行了描述。最后，针对给出的混合调度算法，给出了模拟测试方案，实验证明了该算法能够有效提高共享高速缓存的命中率。

随着多核处理器在实时系统中越来越多的应用，多核平台上的实时调度问题得到了越来越深入的研究。针对实时系统不同的应用需求，需要各种满足不同应用特点的多核实时算法。在此，总结本文一些研究的不足和待继续研究的问题：

（1）针对基于共享缓存的混合调度算法，本文仅对算法进行了模拟实验，在今后的工作中，可以将算法应用到具体的多核实时操作系统中，从而在实践中对算法进行更进一步的分析。

（2）针对全局调度和局部调度的不足，混合调度算法将会得到越来越多的应用，但是，现有的混合调度算法没有固定的调度模型，实现方式也更加复杂，这都需要更进一步的研究。

（3）多核架构的组织方式各种各样，针对不同的多核架构需要研究适应的算法，这样才能更有效的利用多核处理器的处理能力。

（4）多核实时调度算法的研究越来越深入，多核处理器的应用也越来越普遍，可以将现有的优秀的多核实时调度算法应用到实际的系统中。

参考文献

[1] 郑宗汉. 实时系统软件基础. 第 1 版. 北京: 清华大学出版社, 2003, 1-7

[2] Olukotun K, Hammond L, Laudon J. 片上多处理器体系结构. 汪东升, 王海霞, 李鹏译. 第 1 版. 北京: 机械工业出版社, 2008, 1-20

[3] Davis R I, Burns A. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. ACM Computing Surveys, 2011, 43(4): 1-44

[4] Liu C L, Layland J W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM, 1973, 20(1): 46-61

[5] Audsley N C, Burns A, Davis R I, et al. Fixed Priority Preemptive Scheduling: an Historical Perspective. Real Time Systems, 1995, 8(2): 173-198

[6] Sha L, Abdelzaher T, årzén K, et al. Real Time Scheduling Theory: A Historical Perspective. Real Time Systems, 2004, 28(2): 101-155

[7] Liu C L. Scheduling Algorithms for Hard-Real-Time Multiprocessors in a Hard Real-Time Environment. JPL Space Programs Summary, 1969, 37(2): 28-31

[8] Dhall S K, Liu C L. On a Real-Time Scheduling Problem. Operations Research, 1978, 26(1): 127-140

[9] Oh Y, Son S H. Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem. http://beru.univ-brest.fr/~singhoff/cheddar/src/Cheddar-2.1-src /docs/publications/oh93.pdf, 1993

[10] Oh Y, Son S H. Allocating Fixed Priority Periodic Tasks on Multiprocessor Systems. Real Time Systems, 1995, 9(3): 207-239

[11] Burchard A, Liebeherr J, Oh Y, et al. New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems. IEEE Transactions on Computers, 1995, 44(12): 1429-1442

[12] Hwang R, Gen M, Hiroshi K. A Comparison of Multiprocessor Task Scheduling Algorithms with Communication Costs. Computers and Operations Research, 2008, 35(3): 976-993

[13] Omara F A, Arafa M M. Genetic Algorithms for Task Scheduling Problem. Journal of Parallel and Distributed Computing, 2010, 70(1): 13-22

[14] Xhafa F, Abraham A. A compendium of Heuristic Methods for Scheduling in Computational Grids. In: Proceedings of the 10th International Conference on

Intelligent Data Engineering and Automated Learning. Burgos: Springer-Verlag, 2009, 751-758

[15] Phillips C A, Stein C, Eric T, et al. Optimal Time Critical Scheduling via Resource Augmentation (extended abstract). In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing. United States: ACM, 1997, 140-149

[16] Anderson B, Bauah S, Jonsson J. Static-Priority Scheduling on Multiprocessors. In: Proceedings of the 22nd IEEE Real-Time Systems Symposium. UK: IEEE, 2001, 193-202

[17] Goossens J, Funk S, Baruah S K. Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors. Real Time Systems, 2003, 25(2-3):187-205

[18] Baker T P. Multiprocessor EDF and Deadline Monotonic Schedulability Analysis. In: Proceedings of the 24th IEEE Real-Time Systems Symposium. Mexico: IEEE, 2003, 120-129

[19] Baker T P. An Analysis of EDF Scheduling on a Multiprocessor. IEEE Transactions on Parallel and Distributed Systems, 2005, 15(8): 760-768

[20] Baruah S K, Cohen N K, Plaxton C G, et al. Proportionate Progress: A Notion of Fairness in Resource Allocation. Algorithmica, 1996, 15(6): 600-625

[21] Anderson J, Srinivasan A. Early-Release Fair Scheduling. In: Proceedings of the 12th Euromicro Conference on Real-Time Systems. Stockholm: IEEE, 2000, 35-43

[22] Anderson J, Srinivasan A. Mixed Pfair/Erfair Scheduling of Asynchronous Periodic Tasks. Journal of Computer and System Sciences, 2004, 68(1): 157-204

[23] Andersson B, Tovar E. Multiprocessor Scheduling with Few Preemptions. In: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. United States: IEEE Computer Society, 2006, 322-334

[24] Andersson B, Bletsas K, Baruah S. Scheduling Arbitrary-Deadline Sporadic Task Systems on Multiprocessors. In: Proceedings of the 2008 Real-Time Systems Symposium. United States: IEEE Computer Society, 2008, 385-394

[25] Anderson J, Calandrino J, Devi U. Real-Time Scheduling on Multicore Platforms. In: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium. California: IEEE, 2006, 179-190

[26] Anderson J, Calandrino J. Parallel Real-Time Task Scheduling on Multicore Platforms. In: Proceedings of the 27th IEEE Real-Time Systems Symposium. Washington: IEEE, 2006, 89-100

[27] Calandrino J, Anderson J. Cache-Aware Real-Time Scheduling on Multicore Platforms: Heuristics and a Case Study. In: Proceedings of the 2008 Euromicro Conference on RealTime Systems. Prague: IEEE, 2008, 299-308

[28] 黄国睿, 张平, 魏广博. 多核处理器的关键技术及其发展趋势. 计算机工程与设计, 2009, 30(10):2414-2417

[29] 陈国良. 并行计算. 第 3 版. 北京：高等教育出版社, 2011, 1-32

[30] Krishna C M, Shin K G. 实时系统. 戴琼海译. 第 1 版. 北京: 清华大学出版社, 2004, 1-22

[31] 张尧学, 史美林，张高. 计算机操作系统教程. 第 3 版. 北京: 清华大学出版社, 2006, 103-107

[32] Leung J Y T, Whitehead J. On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks. Performance Evaluation, 1982, 2(4): 237-250

[33] Mok A K. Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment. http://dspace.mit.edu/bitstream/handle/1721.1/15670/10728774.pdf?sequence=1, 1983

[34] Coffman E G, Galambos G, Martello S, et al. Bin Packing Approximation Algorithms: Combinational Analysis. http://www.comet.columbia.edu/~egc/webpapers/survbpp5.ps, 1998

[35] Dertouzos M L, Mok A K. Multiprocessor on-line Scheduling of Hard-Real-Time Tasks. IEEE Transactions on Software Engineering, 1989, 15(12): 1497-1506

[36] Andersson B, Baruah S, Jonsson J. Static-Priority Scheduling on Multiprocessors. In: Proceedings of the 22nd IEEE Real-Time Systems Symposium. Washington: IEEE Computer Society, 2001, 193-202

[37] Srinivasan A, Baruah S. Deadline-Based Scheduling of Periodic Task Systems on Multiprocessor. Information Processing Letters, 2002, 84 (2): 93-98

[38] Goossens J, Funk S, Baruah S. Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors. Real-Time Systems, 2003, 25(2-3): 187-205

[39] Baruah S K. Optimal Utilization Bounds for the Fixed-Priority Scheduling of Periodic Task Systems on Identical Multiprocessors. IEEE Transaction Computing, 2004, 53(6): 781-784

[40] Anderson J, Srinivasan A. Pfair scheduling: Beyond Periodic Task Systems. In: Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications. South Korea: IEEE, 2000, 297-306

[41] Lee S K. On-line Multiprocessor Scheduling Algorithms for Real-Time Tasks. In: Proceedings of the IEEE Region 10s Ninth International Conference Theme Frontiers of Computer Technology. Singapore:IEEE, 1994, 2: 607-611

[42] Baker T P, Cirinei M, Bertogna M. EDZL Scheduling Analysis. Real Time Systems, 2008, 40(3): 264-289

[43] Shin I, Easwaran A, Lee I. Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors. In: Proceedings of the 2008 Euromicro Conference on Real-Time Systems. Washington: IEEE Computer Society, 2008, 181-190

[44] Calandrino J M, Anderson J H, Baumberger D P. A Hybrid Real-Time Scheduling Approach for Large-Scale Multicore Platforms. In: Proceedings of the 19th Euromicro Conference on Real-Time Systems. Italy: IEEE Computer Society, 2007, 247-258

[45] López J M, Díaz J L, García D F. Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems. Real-Time Systems, 2004, 28(1): 39-68

[46] 英特尔软件学院教材编写组. 多核多线程技术. 第 1 版. 上海: 上海交通大学出版社, 2011, 1-20

[47] 所光, 杨学军. 多核处理机系统 Cache 管理技术研究现状. 计算机工程与科学, 2010, 32(7): 65-68

[48] Fedorova A, Seltzer M, Small C, et al. Performance of Multithreaded Chip Multiprocessors and Implications for Operating System Design. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference. Anaheim: USENIX Association, 2005, 26-26

[49] Chandra D, Guo F, Kim S, et al. Predicting Inter-Thread Cache Contention on a Multiprocessor Architecture. In: Proceedings of the 11th International Symposium on High-Performance Computer Architecture. Washington: IEEE Computer Society, 2005, 340-351

[50] Fedorova R, Seltzer M, Smith M D. Cache-Fair Thread Scheduling for Multicore Processors. http://www.eecs.harvard.edu/~fedorova/papers/osdi-2006-submis-sion.pdf, 2005

[51] Yiqiang D, Wei Z. Improving the Static Real-Time Scheduling on Multicore Processors by Reducing Worst-case Inter-thread Cache Interferences. In: Proceedings of the 48th Annual Southeast Regional Conference. Mississippi: ACM, 2010, 1-4

[52] Guan N, Stigge M, Yi W, et al. Cache-Aware Scheduling and Analysis for Multicores. http://user.it.uu.se/~yi/pdf-files/emsoft09-yi.pdf, 2010

[53] Devi U, Anderson J. Tardiness Bounds for Global EDF Scheduling on a Multiprocessor. Real-Time Systems, 2008, 38(2): 133-189

[54] Leontyev H, Anderson J. Generalized Tardiness Bounds for Global Multiprocessor Scheduling. In: Proceedings of the 28th IEEE International Real-Time Systems Symposium. Florida: IEEE, 2005, 341-352

附录 A (攻读硕士学位期间发表的学术论文及参与的科研项目)

1. 发表的学术论文:

[1] Yan Wang, Lida huang, Renfa Li, et al. A Shared Cache-Aware Hybrid Real-Time Scheduling on Multicore Platform with Hierarchical Cache. In: 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming (PAAP 2011). Tian jin: IEEE Computer Society, 2011, 208-212

2. 参与的科研项目:

[1] 实时嵌入式操作系统及开发环境. (核高基项目 No. 2009ZX01038-001).

致 谢

时光荏苒，岁月匆匆，转眼又到了毕业的季节，研究生生活就这样结束了。人生中宝贵的青春年华能在湖南大学校园中，在千年学府严谨学风的熏陶中，在众多才华横溢的老师的淳淳教诲中度过，实在感到充实而又荣幸。值此论文完成之际，谨向给予我指导、关心和帮助的老师、朋友、亲人表示最衷心的感谢。

首先，我要衷心感谢我的导师李仁发教授！从论文选题、开题、研究和最终撰写，每一步都倾注了导师的严格要求和悉心指导。李老师渊博的学识和对学术方向高屋建瓴的把握引导了我研究的正确方向，同时，老师博大的胸怀和高尚的情操，严谨的治学态度让我深深敬佩，也是我学习的动力和榜样。李老师求实的科学态度、对事业刻苦钻研及其强烈的责任感和敬业精神都将是我这一生宝贵的精神财富。

特别感谢黄丽达老师和李蕊老师！工作中的许多问题都是在黄老师和李老师的帮助下解决的，正是你们无私的指导，这篇论文才得以完成。黄老师和李老师严谨的治学态度、认真细致的学术作风以及精湛的专业知识都让我受益匪浅，请接受我真诚的谢意。

衷心感谢邓欢老师、李新星老师、彭娜老师、付彬老师、肖玲老师、刘彩苹老师以及实验室所有老师对我的培养和指导，感谢他们对我学习、工作以及生活上的关心和帮助。

衷心感谢本实验室的同学们，特别向袁征、江斌、杨阳、张然、唐鹭、何康、吴虎、龙志超、杨早、付巍玮、阮华斌、钟球、李筱同学表示感谢，在与你们的相处与共同工作当中，我们培养了深厚的感情，大家互相合作，共同探讨学术知识，谢谢你们的热情帮助和指导。

衷心感谢湖南大学信息科学与工程学院各位领导和各位老师的关心与支持。为我们提供了良好的学习环境和实验条件。是你们的辛勤劳动，才使得我们的学习、生活和最后的答辩工作进行的有条不紊。

衷心感谢在百忙中为审阅本文而付出辛勤劳动的各位专家学者。

我还要特别感谢我的父母，感谢你们对我二十多年的养育之恩，你们对我学习上的默默支持和生活上无微不至的照顾，让我没有后顾之忧，是我前进的动力的勇气。特别感谢您们，我亲爱的父母亲！

最后，由衷的感谢所有关心和支持过我的人，祝你们永远健康快乐！

王 艳

2012 年 5 月 9 日