一种降低核间通信开销的调度算法①

韩 乐[1], 陈香兰[1,2], 李 曦[1,2]

[1](中国科学技术大学 计算机科学与科学与技术学院, 合肥 230027)
[2](中国科学技术大学 苏州研究院, 苏州 215123)

摘 要: 近年来, 多核处理器在嵌入式领域得到越来越广泛的应用, 但多核间不可避免的通信开销阻碍了系统性能大幅提升, 因此研究如何降低核间通信开销变得尤为重要. 针对同构多核平台上周期依赖任务, 提出一种降低核间通信开销的任务调度算法并在该基础上进行优化, 通过对部分任务预先调度一个周期, 将周期内任务间的数据依赖转换成周期间的数据依赖, 从而缩短调度长度, 提高系统性能. 对以上算法进行仿真模拟, 并分别在双核和四核平台上进行多组实验. 结果表明: 提出的调度优化算法可以显著降低周期依赖任务核间通信开销, 提高执行效率.

关键词: 核间通信; 周期性任务; 任务调度; 实时系统; 预先调度

Scheduling Algorithm with Communication Overhead Reduction

HAN Le[1], CHEN Xiang-Lan[1,2], LI Xi[1,2]

[1](College of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou 215123, China)

Abstract: Embedded applications in multi-core processors have been widely used in recent years, but the inevitable inter-core communication overhead has blocked system performance continuous improving. So it's important to research on how to reduce inter-core communication overhead. This paper focused on cycle-dependent tasks on homogeneous multicore platforms and proposed an algorithm to reduce schedule length. The algorithm scheduled a few tasks in advance by one cycle so as to the intra-periodic dependency can be converted to inter-periodic dependency. We simulated our algorithm and tested on 2-core and 4-core platforms, and the result showed this method reduced communication overhead and improved system performance relative to the initial algorithm.

Key words: inter-core communication; periodic tasks; task scheduling; real-time; pre-schedule

1 绪论

多核体系结构的广泛应用, 带动了通信、工业控制、多媒体网络等各嵌入式领域在实时性、高性能等方面的发展, 同时也给多核嵌入式系统设计带来了巨大的挑战. 多核操作系统设计面临的最大难题有两个: 一是如何使其上的应用并行化以获得更高的效率; 二是如何对资源合理分配以达到性能最优, 并避免对资源的争用成为系统瓶颈. 衡量实时嵌入式系统的一个重要特性是其时间性能, 而进程调度算法、进程间通信均对系统响应时间有较大影响.

针对于上述系统设计难题, 当前已存在一些相关研究和成果: 文献[1]提出了一种线程级的并行任务调度, 通过线程间的同步降低 L2 cache miss 率, 提高应用并行效率; 文献[2]提出一种同时考虑并行处理和流水线应用的方法对任务进行更合理的分配以提高任务运行效率; 文献[3]提出一种动态任务调度和分配方法解决多核平台的任务映射和执行顺序问题. 这些研究主要通过增加任务执行并行性来提高效率, 但均忽略了核间通信开销的问题.

实时系统的工作负载包括周期性和非周期性任务,

① 收稿时间:2014-01-20;收到修改稿时间:2014-03-17

非周期性任务是依需求动态随机执行, 周期性任务可看成系统的“基本负载”, 并以固定的时间间隔执行. 周期性任务在实时系统中占有重要地位. 在多核环境下, 运行在不同核上的任务之间存在数据依赖, 核间通信不可避免. 通过降低核间通信开销, 可使系统多方面的性能指标, 如: 时间性能、内存使用率、功耗性能均得到显著改善. 因此本文重点讨论多核体系下降低周期性任务间通信开销问题.

多核体系下, 核间通信时间在任务执行时间中占有一定比重, 因此理论上可以通过优化核间通信提高系统执行效率. 目前对核间通信的研究主要集中在以下几方面: 1)改进 MPI 库(Message Passing Interface Library)[4]. MPI 为基于消息传递机制的高性能并行计算编程提供了一套标准. 当前 MPI 库虽然功能较强, 但资源消耗较多, 需要加以改进以适应嵌入式应用; 2)针对于特定的网络连接, 提高通信效率. 如对 mesh 网、grid 网格环境和集群体系结构的任务调度与通信的研究[5,6]; 3)使用更有效的任务分配映射算法, 通过对任务合理分配, 最大程度降低核间通信开销和减少对共享资源的争用. 现有多核映射算法多是基于图映射理论, 而图映射问题是 NP 问题. 基于此理论产生很多启发式算法, DRB 算法和 K-way 图划分算法是其中典型代表[7].

多核通信的重要方式之一是利用总线进行数据传输, 因此本文的工作主要在基于总线的体系结构下进行. 对于总线结构, 已经提出一些基于总线访问策略和总线调度策略的算法: 包括事件触发和时间触发的调度策略[8,9]. 这些技术对总线仲裁提出了很好的解决方案, 但均没有考虑核间通信开销问题. 文献[10]提出一种针对于片上多核流式应用能够完全消除核间通信开销的方法, 但该算法较复杂, 时间复杂度较高.

本文提出一种能够有效降低核间通信开销的任务调度方法: 将计算型任务与通信型任务进行重新调度, 以使部分计算型任务与通信型任务重叠执行, 达到降低核间通信开销的目标. 基本思想是: (1)根据任务间依赖关系进行资源(处理器核)分配. (2)在任务正式启动前, 预先调度最长通信时长的通信型任务及其他一些相关联任务. 在任务正式启动时, 部分所必须的数据传输已经完成, 达到核间通信时间(数据传递时间)被计算任务时间覆盖的目的, 从而提高周期任务执行效率. 总体来说, 本文通过这种资源分配和预先调度的方式提高了总线利用率及任务运行效率.

本文剩余部分组织如下: 第 2 部分建立总线结构的系统模型、周期任务模型; 第 3 部分阐述考虑核间通信的任务分配策略及对其优化的算法; 第 4 部分介绍相关实验以及实验结果分析. 最后得出结论, 说明本文提出的调度算法的合理性和高效性.

2　系统和任务模型

本论文采用一种典型的基于总线的对称多核体系结构, 如图 1 所示, 为有 3 个处理器核 (C_1,C_2,C_3), 一条共享总线, 和一个总线仲裁器的总线结构图. 3 个处理器通过总线进行核间通信, 若同时有多个处理器核(如 core1、core2)同时向总线提出请求时, 则产生总线争用, 此时由总线仲裁负责处理, 任一时刻只能有一个处理器核在总线上进行消息发送, 独占总线. 多核间的通信分为两种: 核间中断、核间数据传输. 两种方式的发送均需要处理器独占总线. 本文研究如何降低核间数据传输所产生的总线争用.

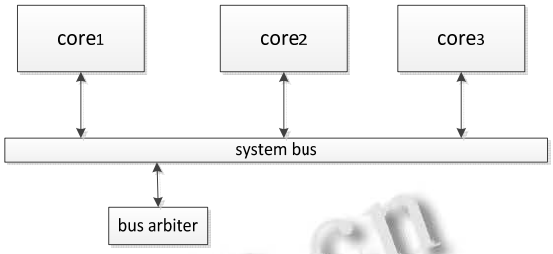

图 1　3-核对称多核结构图

多核上的任务可以划分为计算型任务和通信型任务. 计算型任务在处理器核上完成数据的计算; 通信型任务完成核间数据传输. 本文的前提是已知各周期任务的依赖关系后, 如何对调度进行优化. 任务间依赖关系及通信时间可被建模成 DAG 图(有向无环图), DAG 图可由多种方法得到, 如简单的自学习方式[11]: 依任务提交先后次序, 将每个任务均分配在不同的核上运行: 例如 A 任务在处理器核 C_1 上运行完后, 把相关数据传输到依赖于 A 的、运行在其他核上的任务, 根据 A 任务的完成时间和其他任务的数据接收时间, 容易得到此次核间通信时间. 由于每个任务均知道自己运行所依赖的其他任务, 从而只需要经过一个周期即可得到完整 DAG 图.

一个 DAG 图 G: (T,TR,E). 其中:

$T=(T_1,T_2,.....T_k)$　: 结点集, 每个结点代表一

个周期计算型任务;

TR_{ij}: 结点集, 每一个结点代表一个周期通信型任务, 即描述了从计算型任务 T_i 到 T_j 的数据转换;

$E<T_i,T_j>$: 边集, 定义了结点间的数据依赖关系: 即在同一周期内 T_j 的执行需要 T_i 的结果. 任务间的数据依赖类似于消费者生产者的关系. 设 L 为任务集 T 的执行周期, L 表明从 t 时刻, 所有的任务必须在 $t+L$ 之前完成. 在任一周期 L, 我们用 S_i 表示计算任务 T_i 的开始时间; S_{ij} 为通信任务 TR_{ij} 的开始时间; C_i 为计算任务 T_i 的执行时间; C_{ij} 为通信任务 TR_{ij} 的执行时间.

如图 2 所示为 6 个周期性任务经过自学习过程得到的一种DAG图(不通类型的任务得到的DAG图可能不同). 每个周期内的任务调度均需遵循如图 2 的依赖关系.

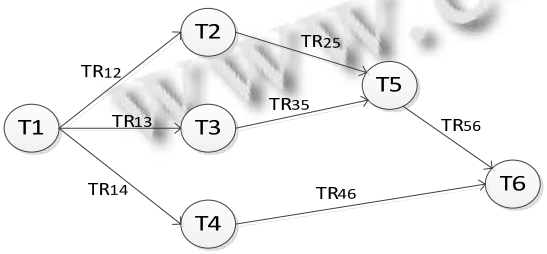

图 2 任务 DAG 图

该图中, 共有 7 条边, 每条边代表两个核之间的数据传输. 每个计算型任务执行时间均为 2 个 cycle: $C_1=C_2=C_3=C_4=C_5=C_6=2$; 通信型任务执行时间 $C_{12}=3$, $C_{13}=2$, $C_{14}=1$, $C_{25}=2$, $C_{35}=1$, $C_{46}=1$, $C_{56}=2$. 若无特殊说明, 本文以下部分涉及到的示例DAG图均指本图.

3 静态调度算法

3.1 一种简单的静态多核调度算法

对于任一个给定DAG图的静态调度, 既包括对任务的处理器分配和对任务开始时间的指定, 又要求任务执行次序必须遵循任务之间的依赖关系.

一种最简单的达到负载均衡的静态调度算法为 Cyclic 算法. 该算法的思想是: 以轮询的方式周期性的分配每个任务到每个处理器核上. 例如: 16 个任务分配到 8 个核上, 分配方式为: 0 号任务和 8 号任务分配到 0 核上, 1 号和 9 号任务分配到 1 核上.....依次类推.

图 3 是以图 2 的 DAG 图为例, 按 Cyclic 算法得出的任务调度时序图, 此时调度长度(任务集 T 中所有任务执行一个周期所需时间)为 16cycle.

Cyclic 算法只是简单的对任务进行轮询分配, 保证了各个核上较好的负载平衡. 但这种分配策略完全没有考虑任务间依赖与核间通信问题, 对于核间通信频繁的两个任务(存在数据依赖), 若恰分配到两个核上运行, 相对于分配到同一个核上, 会造成调度长度增加, 并行任务执行效率低下.

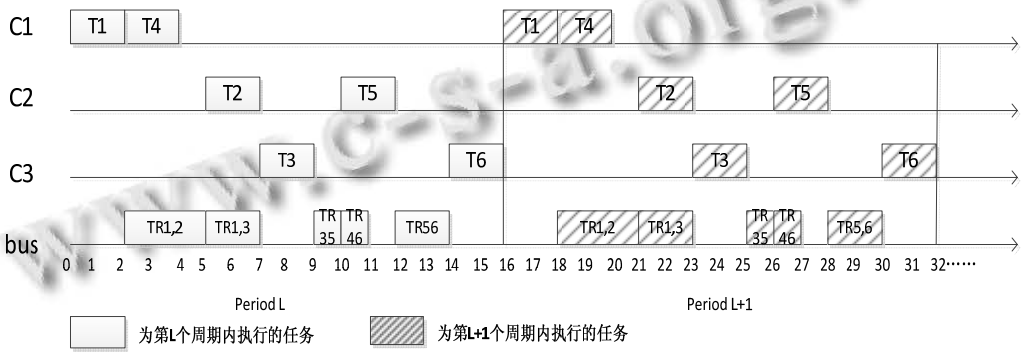

图 3 Cyclic 算法调度时序图

3.2 简单基于核间通信的静态调度算法

针对 Cyclic 算法的不足, 我们提出简单的基于核间通信的静态 SSC 调度算法(simple schedule algorithm based on communication), 其思想是: 根据已有的任务 DAG 图, 对于同时有多个依赖关系的任务, 依核间通信时长, 尽可能多的将核间通信时长较长的两个任务(通信开销用通信时长来衡量)分配到同一个核上串行执行, 这样可以最大程度的减小核间通信开销; 否则若使存在较长核间通信时长的两个任务分配到不同核上运行, 因核间数据传输需要顺序使用总线, 势必造

成调度长度变长.

我们提出的 SSC 算法描述如下:

1)当分配某一任务 T 时, 若没有其他任务与 T 同时依赖同一个任务, 则对 T 按照 Cyclic 的方式以轮询进行核分配;

2)当 T 与其他任务存在依赖关系时, 可分为如下两种:

若 T 同时依赖于 M、N.....多个任务, 如图 4.

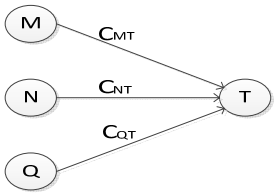

图 4　两种依赖关系(1)

此时找出多个通信型任务间的最长通信时间, 将 T 分配到与 M 为同一个核上.

若存在其他任务 P、R.....与 T 同时依赖某一个任务 A, 如图 5.

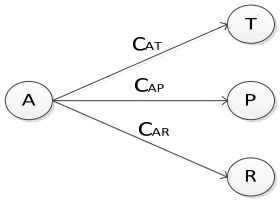

图 5　两种依赖关系(2)

C_{MT} 此时找出多个通信型任务间的最长通信时间 C_{AP}, 把 P 分配到与 A 同一个核上去运行, 其他任务则按照 Cyclic 方式进行分配.

当处理 T 任务上述两种情况同时存在时, 由于任务依DAG图从前向后以轮询的方式分配, 则优先按照情形 1 处理.

该算法描述如下:

Algorithm SSC

Input: tasks dependency relation: DAG $G=(V,TR,E)$ and the processor cores number: N

Output: assignment map from tasks to cores

Begin

1) assign the first task T_1 on core C_1;

/*把 T_1 分配到 C_1 上执行*/

2) find all the tasks which are independent of T_1 ,and

assign them to other cores in round-robin method;

/*把所有与 T_1 无依赖关系的任务以轮询的方式分配到各个核上运行*/

3) for (T_1 to T_s)

/*以轮询的方式处理每一个任务*/

a. if (T_i is dependent of several tasks)

/*若当前处理任务 T_i 依赖于多个任务*/

traverse all the inter-core communication time which T_i are dependent to find the max value and relative task T_j ,and assign T_i to the core the same with T_j;

/*找最长核间通信的任务, 并分配到同一核*/

b. if (T_i and other tasks are dependent of the same core T_a)

/*当存在多个任务与当前处理任务依赖于同一个任务时*/

traverse all the inter-core communication time to find the max one, and assign the dependent task T_z (may be T_i) to the core the same with T_a;

/*找最长核间通信的任务, 并分配到同一核*/

c. assign T_i to the core the same with T_a;

/*若 T_i 只依赖于 T_a, 只存在一个依赖关系的两个核分配到同一个核上执行*/

End

该算法只需遍历一次任务调度时序图即可完成, 时间复杂度为 O(n).

图 6 为以图 2 的 DAG 图为输入, 依据 SSC 算法得出的任务调度时序图. 首先分配 T_1 到 C_1, 由于此时 $TR_{12} > TR_{13} > TR_{14}$, 所以将 T_2 分配到与 T_1 同一个核 C_1 上运行, T_3、T_4 依次分配到 C_2、C_3 上运行, 当 T_2、T_3 均运行完毕后, 再由于 $TR_{25} > TR_{35}$, T_2 与 T_5 之间的核间通信时长较长, 则将 T_5 分配到 C_1 上去运行. 以此类推, 可以看出每个周期的调度的长度为 11cycle, 较使用 Cyclic 算法减少 5 个 cycle, 当周期数增加时, 使用 SSC 调度算法会大幅降低任务执行时间.

基于核间通信的静态任务调度算法(SSC)在每次对任务进行处理器核分配时, 仅试图将核间通信开销较大的两个任务分配到同一个核上去运行, 避免了 Cyclic 算法中大量核间通信开销, 但运行在不同核上存在关联关系的任务间的通信开销仍不可避免, 任务

调度长度仍然较长.

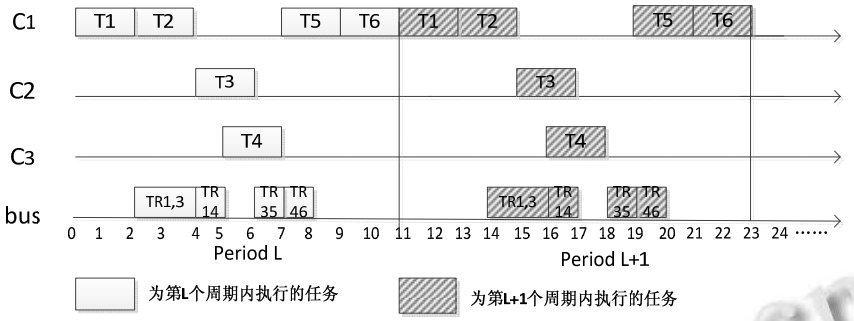

图 6 SSC 算法调度时序图

3.3 优化的基于核间通信的静态调度算法

为进一步缩短调度长度, 我们将流水线思想运用到 SSC 算法上, 进一步削弱运行在不同核上的任务间相互制约关系, 使任务间的并行执行程度更高, 进而提出优化的基于核间通信的任务静态调度 OSSC 算法(optimized simple schedule algorithm based on communication). 通过把不同周期内的任务进行重组织、重新调度, 进一步降低核间通信造成的任务执行延迟. 具体方法如下:

依据 SSC 算法得到的调度时序图中总线的占用情况, 得出依赖于各个计算型任务 T_1、T_2 、T_3 ..... T_k 的占用总线时间最长的通信型任务(TR_{1x} 、 TR_{2x} 、 TR_{3x} …… TR_{kx}). 任意 TR_{ix} 可能只由一个通信型任务 TR_{1x} TR_{is} 或多个连续的通信型任务 $TR_{is}+TR_{ip}+TR_{iq}$..... 组成. 为下述描述方便, 该通信型任务用 TR_{ix} 表示.

得出(TR_{1x}、TR_{2x}、TR_{3x} …… TR_{kx})中占用总线时长最长的任务 TR_{mx}, 进而找到依赖 TR_{mx} 的所有计算型任务;

预先调度周期内的部分任务: 从开始任务 T_1 到最长通信型任务(TR_{mx})之间的所有任务;

从下一周期开始, 依赖最长核间数据传输(TR_m)的任务可依赖上一周期产生的数据传输结果进而提前执行, 从而以后的每一周期计算型任务与核间通信型任务覆盖执行, 达到类似于流水线的效果, 进而缩短调度长度.

仍以图 3 所示 DAG 图为例, 从 SSC 算法得出的任务调度时序图(图 6)可看出, 占用总线时间最长的是与 T_1 相关的 $TR_{13}+TR_{14}$, 因此通过在第 L-1 个周期内对 T_1、T_2、TR_{13}、TR_{14}(对 TR 类型任务执行一个周期表示将 T_1、T_2 产生的数据传送到需要的处理器核上)预先调度一个周期(5cycle, 实际小于一个周期的时长), 从下一周期 L 开始, T_3 和 T_4 可以得到第(L-1)个周期 TR_{13} 和 TR_{14} 的数据, 而总线上同时可以运行第 L 周期的 TR_{13} 和 TR_{14}, 处理器核 C_1 上也运行着第 L 个周期内执行的任务 T_1 和 T_2. 进而每当 T_3 和 T_4 需要执行时, 所需的 T_1 的数据都是就绪的, 而不需要等待. 改进的任务调度时序图如图 7 所示. 此时单周期调度长度从 11cycle 变成了 10cycle, 多周期任务中, 一个 cycle 的优化是相当重要的. 而必须注意到的一点是: 通过 OSSC 算法进行优化效率提升的幅度与任务的 DAG 图有直接关系, 不同 DAG 图优化的效果差距可能较大.

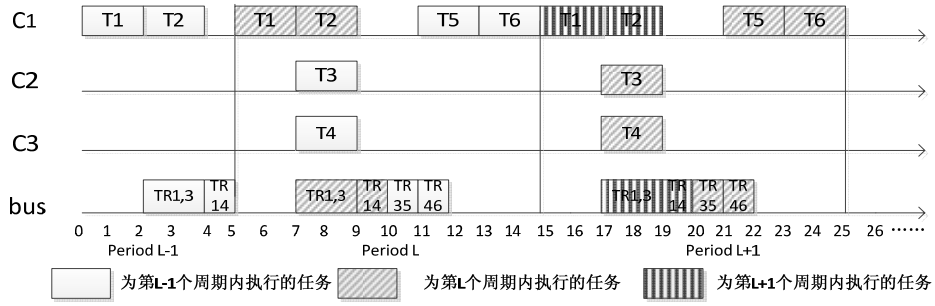

图 7 OSSC 算法调度时序图

基于核间通信的任务调度优化算法描述:

Algorithm OSSC

Input: tasks dependency relation: DAG *G=(V,TR,E)* and the processor cores number: *N*

Output**:** assignment map from tasks to cores

Begin

1)According the timing diagram of algorithm SSC, get the maximum occupancy bus communication task ;
/*依时序图得到依赖于每个计算型任务的最长占用总线的通信型任务*/

2)Get the max occupancy task among(TR_{1x} 、 TR_{2x} ,.....) and the relative task TR_{mx} (maybe several tasks);
/*得到最长占用总线计算型任务 TR_{mx} 和与之相关联的任务*/

3)Pre-scheduling a period from T_1 to TR_{mx}
/*预先调度从 T_1 到 TR_{mx} 的所有任务*/

4)From next period the computing tasks related to TR_{mx} can be scheduled ahead;
/*依赖于 TR_{mx} 的计算任务可以提前执行*/

End

该算法只需遍历一次任务调度时序图即可完成, 时间复杂度为 O(n).

总体来说, OSSC 算法较 SSC 算法主要提高在使周期间部分任务执行重叠, 提高时间利用率. OSSC 算法削弱了依赖占用总线时间最长的通信型任务的各计算型任务之间的数据依赖, 达到即使任务间存在依赖关系, 仍可并行执行的目的, 进一步减少了核间通信时间、缩短了任务调度长度.

4　实验环境及结果分析

本节介绍核间通信相关调度算法的实验. 实验环境配置如下:

表 1　实验环境配置

实验配置	参数
CPU	Intel Core 双核/四核
内存	2G
操作系统	Linux2.6.38
编译环境	JDK、Eclipse

本文使用 Java 多线程模拟多任务环境, 并实现 Cyclic、SSC、OSSC 算法. 通过在双核/四核平台上运行来模拟多任务在不同数目的处理器核下运行情况. 若两个任务(*A*、*B*)存在依赖关系, 并运行在相同的核上, 则串行执行这两个任务; 而若(*A*、*B*)需要分配到不同核上运行, 则在 *A* 与 *B* 之间再增加一个进行数据传输的通信任务 C, 再串行执行这三个任务. 当两个任务(*A*、*B*)不存在依赖关系, 若应将这两个任务分配到同一个核上运行, 则以串行执行; 而若应将其分配到不同核上运行, 则 *A*、*B* 之间无等待关系, 直接并行执行.

我们使用 TGFF[12] benchmark 产生随机的任务依赖关系图. TGFF 是一个软件包, 应用其可以产生伪随机的多任务依赖图. 我们设定每个 cycle 为 3 毫秒, 每个计算型任务和每个通信型任务运行完成需要若干 cycle.

为了验证 SSC, OSSC 算法在不同平台上性能及算法随 DAG 图中任务数、边数变化情况的影响, 我们分别在双核和四核平台上进行多组实验. 举例说明, 实验一在双核下分别比较 5 个任务、9 个任务、21 个任务在不同边数(依赖关系个数不同)时, 使用 Cyclic、SSC、OSSC 三个算法的调度长度. 实验中每组数据均是运行 10 次的平均结果. 结果对比图如图 8 所示.

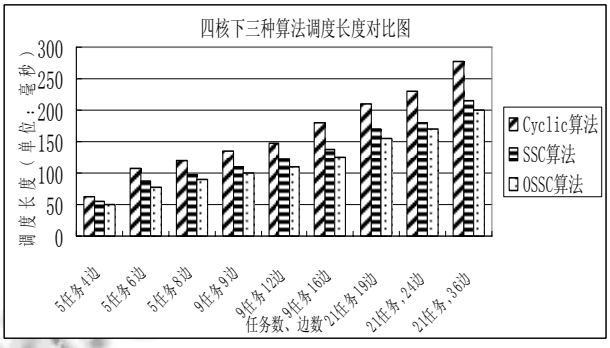

图 8　双核下三种调度算法调度长度对比图

实验二在四核下进行如上相同参数的测试. 结果对比图如图 9 所示.

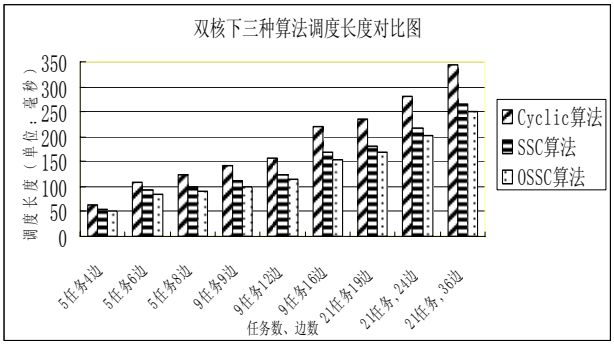

图 9　四核下三种调度算法调度长度对比图

从图 8 和图 9 可得出以下结论: 1)从总体趋势上来看, SSC 算法较 Cyclic 算法效率有显著提升; OSSC 算法较 SSC 算法效率又有一定程度提升. 2)当任务数一定, 边数越多, 即核间通信任务越多, 任一算法的调度长度均随着核间通信任务增多而增长. 这是由于边数增加, 任务间的数据依赖性增强, 任务运行时相互等待的情况就会增加, 故而增加了调度长度. 3)随着任务数的增加, OSSC 算法较 SSC 算法的优化效率有所降低, 这是由于任务数增加, 完成周期内所有任务时间必增加, 而 OSSC 算法每次只处理依赖最长占用总线通信任务的计算型任务, 因此优化效率有所下降. 4.在双核或四核平台上, SSC 算法和 OSSC 算法均能显著提高任务运行效率.

虽然任务的总执行时间与任务的 DAG 图直接相关, 我们无法计算 SSC、OSSC 较传统算法的效率提升精确值, 但本文使用 TGFF benchmark 经过多次模拟实验, 得出 SSC、OSSC 近似的效率提升. 经计算, 在双核下, SSC 算法较 Cyclic 算法效率提高 19.8%; OSSC 算法较 SSC 算法相效率提高 7.8%. 在四核下, SSC 算法较 Cyclic 算法效率提高 19.2%; OSSC 算法较 SSC 算法相效率提高 8.5%. 平均 SSC 算法较 Cyclic 算法效率提高 19.5%; OSSC 算法较 SSC 算法效率提高 8.2%.

由以上两组实验可以看出, 本文提出的优化的基于核间通信的静态任务调度算法在一定程度上降低了核间通信开销, 即提高了任务执行效率.

5 结语

核间通信开销不可避免的存在于多核平台中. 针对于多核下周期性任务调度, 为了减少核间通信开销, 本文主要提出两方面优化策略: 一是根据任务在不同核上的通信时长对任务进行处理器核的分配, 将需要较长通信时长的计算型任务分配到同一个核上运行; 二是根据任务调度时序图中找出占用总线时长最长的通信型任务, 通过预先调度周期内的部分任务, 将部分核间数据传输提前完成, 来达到核间通信任务与依赖于此的计算型任务并行执行, 进而缩短调度长度. 本文以 Cyclic 算法为基础, 进行上述两方面的优化, 通过对比实验表明任务执行效率得到一定的提升. 我们下一步的工作是要证明本文提出的任务重调度和预先调度策略对于其他复杂调度算法(本文基于 Cyclic 算法)同样具有良好的适用性.

参考文献

1 Anderson JH, Calandrino JM. Parallel real-time task scheduling on multicore platforms. Real-Time Systems Symposium. 2006. 89–100.

2 Xu RB, Melhem R, Mosse D. Energy-aware scheduling for streaming applications on chip multiprocessors. Real-Time Systems Symposium.Tucson, AZ. 2007. 25–38.

3 Chen YS, Shih CS, Kuo TW. Dynamic task scheduling and processing element allocation for multi-function SoCs. Real Time and Embedded Technology and Applications Symposium. Bellevue,WA. 2007. 81–90.

4 Chen H, Chen WG, Huang J, Bob R, Kuhn H. MPIPP: An automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. Proc. of the 20th Annul International Conference on Supercom-putting. New York. 2006. 353–360.

5 Chou CL, Marculescu R. User-Aware Dynamic Task Allocation in Networks-on-Chip. Design, Automation and Test in Europe. Munich: 2008: 1232–1237.

6 Zhu Q, Agrawal G. Resource allocation for distributed streaming applications. Parallel Processing, 2008. ICPP'08. 37th Int. Con. Portland. 2008. 414–421.

7 Diaz J, Petit J, Serna M. A survey of graph layout problems. Journal ACM Computing Surveys, 2002, 34(3): 313–356.

8 Pop P, Eles P, Peng Z, Pop T. Analysis and optimization of distributed real-time embedded systems. Proc. of the 41st Annual Design Automation Con. New York. 2006. 593–625.

9 Pop T, Eles P, Peng Z. Design optimization of mixed time/event-triggered distributed embedded systems. Proc. of the 1st IEEE/ACM/IFIP Int. Con. on Hardware/Software Codesign and System Synthesis. New York. 200. 83–89.

10 Wang Y, Liu D, Qin ZW, Shao ZL. Optimally removing intercore communication overhead for streaming applications on MPSoCs. IEEE Trans. on Computers. IEEE Computer Society, 2012, 62(2): 336–350.

11 Gleeson MJ, Hickey J, Linehan P, Loughran K. Network switch with self-learning routing facility. US 09/544, 542, 2004-07-13.

12 Dick RP, Rhodes DL, Wolf W. TGFF: Task graphs for free. Proc. of the 6th International Workshop on Hardware/software Codesign. Washington, DC. 1998. 97–101.