一种实时异构嵌入式系统的任务调度算法*

邱卫东+, 陈 燕, 李洁萍, 彭澄廉

(复旦大学 计算机与信息技术系,上海 200433)

A Task Scheduling Algorithm for Real-Time Heterogeneous Embedded Systems

QIU Wei-Dong+, CHEN Yan, LI Jie-Ping, PENG Cheng-Lian

(Department of Computer and Information Technology, Fudan University, Shanghai 200433, China)

+ Corresponding author: Phn: +86-21-65643192, E-mail: 021021069@fudan.edu.cn, http://www.fudan.edu.cn

Received 2003-03-31; Accepted 2003-11-18

Qiu WD, Chen Y, Li JP, Peng CL. A task scheduling algorithm for real-time heterogeneous embedded systems. *Journal of Software*, 2004,15(4):504~511.

http://www.jos.org.cn/1000-9825/15/504.htm

Abstract: Heterogeneous computing environments have been widely used in real-time embedded systems. Efficient task scheduling is essential for achieving high performance in the synthesis of embedded systems. The problem has been proved to be NP-complete and mainly heuristic algorithms which often have room for improvement exist. In this paper, an algorithm called the dynamic BLevel first (DBLF) has been presented. The DBLF algorithm selects the ready task with a maximum Blevel (n_i) at each step and assigns the selected task to a processor in an insertion mode. The task is assigned to the suitable processor that satisfies the precedence sequence and has the minimum earliest-finish-time (EFT) of the task. When the EFT costs are equal, the task is firstly assigned to the processor which has the least utilization. Compared with the related work, the result shows that the DBLF algorithm significantly surpasses the previous approaches in scheduling length.

Key words: heterogeneous system; list scheduling; scheduling length; dynamic critical path; communication resource access; earliest finish time

摘 要: 异构分布式系统已被广泛应用在实时嵌入式系统中,而调度算法是在进行嵌入式系统综合时,确保系统实现性能目标的一个关键问题,这是一个 NP-完全问题.现有的算法主要是启发式算法,性能还有待提高.提出了一个异构分布式系统的动态 BLevel 优先(dynamic BLevel first,简称 DBLF)算法,算法选择就绪任务中动态 BLevel 值最大的任务进行调度,用插入法为任务分配处理器,遵循以下 3 个插入原则:满足任务先后顺序关系;任务的最早完成时间(earliest-finish-time,简称 EFT)最小;在 EFT 相等时,优先分配到利用率较低的处理器上.与现有算法比较可以看出,DBLF 算法可以有效降低调度长度.

关键词: 异构系统;列表调度;调度长度;动态关键路径;通信资源访问;最早完成时间

* Supported by the National Natural Science Foundation of China under Grant Nos.69873010, 69873010 (国家自然科学基金)

作者简介: **邱卫东**(1972－),女,江苏阜宁人,博士生,主要研究领域为实时嵌入式系统协同设计;**陈燕**(1972－),女,博士生,主要研究领域为实时嵌入式系统设计,系统级建模与验证;**李洁萍**(1979－),女,工程师,主要研究领域为实时嵌入式系统测试技术;**彭澄廉**(1941－),男,教授,博士生导师,主要研究领域为数字系统计算机辅助设计,容错计算与测试技术,嵌入式系统的设计技术.

中图法分类号: TP316 文献标识码: A

实时嵌入式系统中的任务都有严格的时间限制,在设计时不但要求系统的功能正确性,而且要求计算这些功能结果的时间要及时,因此时间性能是实时嵌入式系统的设计中最重要的性能要求;而且,随着应用范围的扩大和复杂度的提高,异构分布式系统已被广泛应用在实时嵌入式系统中,对于时间性能要求的分析也越来越困难.

在进行实时嵌入式系统协同综合时,需要对所有硬件或软件处理单元(processing elements,简称 PE)上执行的任务和任务间的通信进行有效的调度[1],这是整个系统满足性能要求的关键.调度问题通常包含两个方面:一是将任务分配到合适的硬件或软件处理单元上去,同时考虑通信资源的可用性;二是决定每个处理单元或通信资源上的任务执行顺序.

调度问题通常用任务图来描述[1].这种任务图是一种加权有向无环图(directed acyclic graph,简称 DAG),节点表示任务,边反映了任务间的数据依赖.

目前对多处理器系统的任务调度问题研究大多集中在同构系统上,而简化后的同构系统调度问题也是NP-完全问题[2],所以在实际应用中,常常采用启发式算法解决调度问题.主要的启发式算法有列表调度(list-scheduling)算法[3,5~9]和遗传算法[4],而且列表调度算法与遗传算法相比更加实用,调度时间也较短.列表调度算法包含两个步骤:第 1 步是根据优先级进行任务选择,第 2 步是分配处理器,即将第 1 步选择的任务调度到启发函数最小的处理器上去.已有的同构多处理器系统的列表调度算法有 Modified Critical Path(MCP)[5], Dynamic Critical Path(DCP)[3]等.其中 DCP 算法与其他算法相比效率较高[3],但是 DCP 算法存在以下缺陷:(1) 只有当节点在动态关键路径上时才引入新的处理器,但是当任务图中只有一条绝对关键路径,且已经被分配到一个处理器上时,其他一些非关键路径上的任务因为不能引入新处理器而必须被分配到同一个处理器上,从而使调度长度过长,即任务不在关键路径上时,也可能需引入新处理器以保证并行.(2) 任务 n_i 在处理器 p_j 上不可分配时,延迟处理器 p_j 上的某些任务进行插入分配,但是为了避免造成某些已调度任务的执行顺序冲突,延迟的可行性会很小.而且,DCP 算法研究的是同构多处理器系统调度算法,所以调度时仅以任务的最早开始时间作为分配处理器的标准.

在异构系统上任务调度问题研究相对较少[6,7,9],已有的算法如 Dynamic-Level Scheduling(DLS)[6],MH 算法[7]和 HEFT 算法、CPOP 算法[9]也属于列表调度算法.其中,HEFT 算法有较高的效率[9],算法机制如下:第 1 步,选取节点时根据任务执行时间的初始值静态规定 rank_u 的值(任务 n_i 的 rank_u 值等于任务 n_i 到出口任务的最长路径长度),并按任务 rank_u 的非递增顺序选取任务,但是由于 rank_u 的值在算法开始时静态规定,不能动态反映调度后任务图的变化,因此不能保证调度过程中总能选择具有最大的 rank_u 值的任务;第 2 步,选取处理器时的标准是任务的最早完成时间,不考虑系统的拓扑连接和通信资源的可用性,选择标准还需要改进.

上述调度算法大多不考虑多处理器系统的互连方式,仅假定系统中的通信设备可以无竞争地随时访问,但是,在实时嵌入式系统综合时,调度必须考虑系统的通信行为,才能得到更精确的调度结果,从而使系统具有更好的时间特性.本文对实时异构嵌入式系统的任务分配和调度算法进行了研究,利用动态关键路径概念[3],在优化通信资源访问的基础上,提出了一个新的异构分布式系统的 DBLF(dynamic-BLevel-first,动态最大 BLevel 任务优先)算法.

1 问题描述

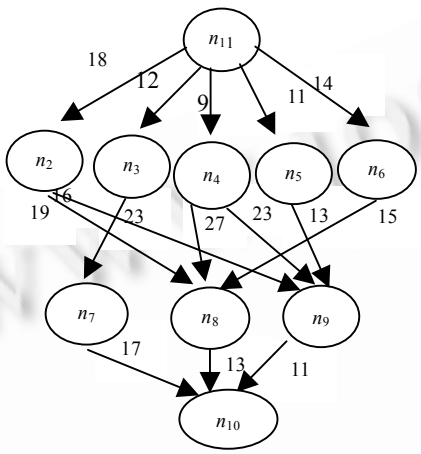
调度系统模型包含应用任务图、运行环境以及应用的任务截止期.任务图是加权有向图 DAG,定义如下:

$$G=(V,E,SourceNode,SinkNode).$$

其中节点集 V 是应用中的任务($|V|=n$),任务执行时间为 R_i.$E=V\times V$ 是图中的边集($|E|=m$),边(n_i,n_j)限定任务的数据依赖关系,任务 n_i 是前驱,任务 n_j 是后继,任务执行的数据依赖关系决定了任务执行的顺序关系:只有前驱任务执行完毕,后继任务才可以执行;边上的权 $C_{i,j}$ 表示任务 n_i 和任务 n_j 之间分配在不同处理单元上的通信时间

(inter_processor communication,简称 IPC),如果前驱任务和后继任务分配到同一个处理单元上,则两个任务之间的通信时间 $C_{i,j}$ 为 0.图中没有前驱任务的任务是图的入口任务 *SourceNode*,没有后继任务的任务是出口任务 *SinkNode*,本文中的算法规定任务图中必须只包含惟一的入口和出口任务.

运行环境包含多个处理单元以及处理单元之间的拓扑连接.处理单元集 Q 中的处理单元可以是通用的处理器 CPUs 或硬件 ASICs 等,任务集 V 在处理器单元 Q 上的执行矩阵 $W=V\times Q,w_{i,j}$ 表示任务 n_i 在处理单元 p_j 上的估计执行时间(如图 1 所示).处理单元之间的拓扑连接支持处理单元间的通信,假定系统中通信设备是专用通信硬件,即通信和计算可以并行进行.设通信设备集合为 B,处理单元集 Q 中处理单元之间的通信设备映射函数为 $F(P_i,P_j)$,规定在 $i=j$ 时,$F(P_i,P_j)=0$,其他函数值由处理单元之间的拓扑连接决定.如图 2(a)所示,$B=\{B1,B2,B3\}$,$F(P_1,P_2)=F(P_2,P_1)=B1,F(P_2,P_3)=F(P_3,P_2)=B2,F(P_1,P_3)=F(P_3,P_1)=B3$.而图 2(b)中,$B=\{B1\}$,且如果 $i\neq j$,则 $F(P_i,P_j)=B1$.



Task	P_1	P_2	P_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	21	7	16

Fig.1 A sample task graph with 10 tasks in the Fig.3 of Ref.[9] and computation costs
图 1 文献[9]的图 3 中包含 10 个任务的任务图和任务执行时间表

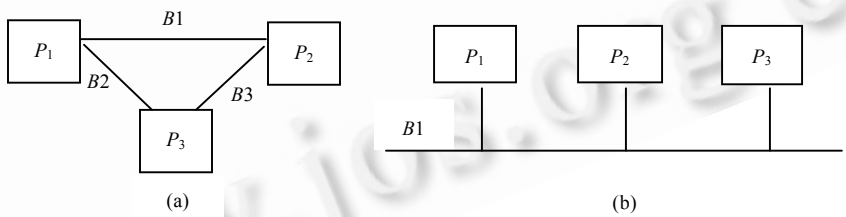


Fig.2 Two interconnect architectures with 3 processing elements
图 2 含 3 个处理单元的目标体系结构两种互联方式

假定运行环境中对一个通信设备的访问是串行方式,通信设备之间则可以并行访问.如图 2(a)所示的系统,数据可以同时在处理单元 P_1 和 P_2 之间通过 $B1$ 交换数据、处理单元 P_1 和 P_3 之间通过 $B2$ 交换数据、处理单元 P_3 和 P_2 之间通过 $B3$ 交换数据.而如图 2(b)所示,系统的拓扑连接,在某一特定时间,只能由 P_1,P_2,P_3 中任两个处理单元通过 $B1$ 进行数据交换,其他通信行为必须等待.

应用的任务截止期(deadline)表示完成任务图的时间期限,用常数表示,如图 1 所示的任务截止期可以为 deadline=80.嵌入式系统已广泛应用于智能设备、智能仪器仪表、信息电器、移动计算、通信等领域:一个典型的第三代移动通信系统,硬件平台的构造包含多处理器(包括 MCU 和 DSP):用于通信的多个 MPC82XX、用于控制的 MPC8XX 或 MPC75X、用于数据处理的多个 DSP(如 TMS320C6X),运行在这些处理单元上的任务必须进行合理的调度才能满足时间限制.

下面给出一些基本定义.

定义 1. 任务执行时间的初始值等于该任务在所有处理器上的平均执行时间.

$$Ini(R_i)=\underset{q\in Q}{average}\{w_{i,q}\} \tag{1}$$

当任务 n_i 调度后被分配到处理器 p_k 时,$R_i=w_{i,k}$.

定义 2. $tlevel(n_i)$是从任务图的入口任务到任务 n_i 的最长路径的长度(不包含任务 n_i 本身的执行时间),设 $pred(n_i)$是任务 n_i 的直接前驱集:

$$\left.\begin{array}{l} tlevel(SourceNode)=R_{SourceNode} \\ tlevel(n_i)=\max\limits_{n_j\in pred(n_i)}\{tlevel(n_j)+C_{j,i}\} \end{array}\right\} \tag{2}$$

$blevel(n_i)$是从任务 n_i 到出口任务的最长路径的长度(包含任务 n_i 本身的执行时间).$succ(n_i)$是任务 n_i 的直接后继集.

$$\left.\begin{array}{l} blevel(Sinknode)=R_{SinkNode} \\ blevel(n_i)=\max\limits_{n_j\in succ(n_i)}\{blevel(n_j)+R_i+C_{i,j}\} \end{array}\right\} \tag{3}$$

定义 3. 关键路径(critical path):任务图中的最长路径,是组成图中最大的执行代价和通信代价的任务集和边集.关键路径长度$|L_{CP}|$定义为

$$\left|L_{CP}\right|=\max_{n_i\in V}\{tlevel(n_i)+blevel(n_i)\} \tag{4}$$

动态关键路径:关键路径在调度过程中不是静态不变的,因为在两个任务分配到同一个处理器上后,两个任务间的通信代价降为 0,从而引起图中路径的长度发生改变;另一方面,在异质系统中,由于任务的初始执行时间等于处理器间的平均执行时间,在任务分配到具体的处理器后,任务的执行时间也会改变,从而引起关键路径发生变化,所以每一个调度步骤后,由于通信量和任务执行值改变,任务图发生了改变,动态关键路径就是当前调度步骤后重新计算的关键路径.

定义 4. 就绪任务(ready tasks):如果任务 n_i 的所有前驱任务都已经执行完毕或者没有前驱任务,则任务 n_i 属于就绪任务.任务图调度前的就绪任务集合中仅包含入口任务{*SourceNode*}.

定义 5. 调度长度(schedule length):任务图中的所有任务到调度完毕后,出口任务的调度完成时间.

2 DBLF 算法

在进行异构系统的任务图调度之前,每个任务的执行时间初始值由公式(1)决定.DBLF 算法属于列表调度算法,算法包含以下两个步骤:

第 1 步.选择任务,原则是选取当前任务图的就绪任务中,具有最大动态 *blevel* 值的任务进行调度.如果存在多个任务具有最大的 *blevel* 值,则随机选取以降低算法复杂度.

第 2 步.分配处理单元,如果是硬件实现的任务,由于硬件可以并发运行,而且任务在硬件上运行的时钟周期数不会随环境变化而改变,所以硬件任务直接分配给硬件 PE;如果是软件实现的任务,将任务插入到所有可运行的处理器中,并选择具有最小累计 EFT(earliest finish time)值的处理器作为最后任务分配目标.在插入分配时遵循以下原则:

原则 I. 存在可插入时间段.设 $schedule(n_k,p_j)$表示任务 n_k 分配在处理器 p_j 的开始执行时间,在处理器 p_j 上分配的任务 n_1~n_m 按开始时间由低到高顺序排列,且设定 $n_0=0;n_{m+1}=\infty$.可插入时间为

$$\left.\begin{array}{l} avail(n_i,p_j)=\max\{schedule(n_k,p_j)+R_k\},\text{其中} \\ k=\min\limits_{0\le l\le m}\{schedule(n_{l+1},p_j)-schedule(n_l,p_j)-R_l>=w(n_i,p_j)\} \end{array}\right\} \tag{5}$$

因为通信设备的访问方式与处理器的访问方式相似,所以通信行为 $C_{i,j}$ 在通信设备 Bi 上的可插入时间段 $avail(C_{ij},Bi)$的计算公式与公式(5)相似.

原则 II. 满足前后顺序关系.设 $insert(n_i,p_j)$表示将任务 n_i 插入到处理器 p_j 的开始时间,$insert(n_i,p_j)$的值由以下 3 个因素决定:任务 n_i 的直接前驱任务的最早完成时间、任务 n_i 与其直接前驱任务之间的通信完成时间

(communication-finish-time,简称 CFT)(公式(6))、处理器 p_j 上的可插入时间段(公式(5)).

$$CFT(c_{pi}) = insert(c_{p,i}, F(P_{n_p}, P_{n_i})) = \max\{EFT(n_p), avail(c_{p,i}, F(P_{n_p}, P_{n_i}))\} \quad (6)$$

$$insert(n_i, p_j) = \max\{\max_{n_p \in pred(n_i)}\{EFT(n_p), r_1 \times CFT(c_{p_i}) + c_{p,i})\}, avail(n_i, p_j)\} \quad (7)$$

其中 P_{n_i}, P_{n_p} 表示任务 n_i,n_p 分配的处理器,如果 $P_{n_i} = P_{n_p}$,则 r_1=0,否则 r_1=1.

原则 III. 向前看一步.设 n_c 是任务 n_i 的 L_{CP} 最大的后继 n_c,n_i 分配到满足公式(9)的处理器.

$$EFT(n_i, p_j) = insert(n_i, p_j) + w(n_i, p_j) + insert(n_c, p_j) + w(n_c, p_j) \quad (8)$$

$$schedule(n_i, p_j) = \min_{p_j \in Q}\{EFT(n_i, p_j)\} \quad (9)$$

向前看一步的原则在调度任务 n_i 时,能避免仅仅以当前任务的最早完成时间为选择处理器标准时,造成其后继任务的最早完成时间太大.

原则 IV. 平均负载.如果存在多个处理器有相同的 cost 值,则在所有处理器之间平均利用率,即将任务优先调度到任务较少的处理器上.如果任务图中存在多条关键路径,这种方案可以潜在地将不同的关键路径分配在不同的处理器上,从而提高并行并降低调度长度;另外,还可以避免在最终的分配方案中,系统中某些处理器过于繁忙,另一些处理器则相对空闲.

算法. DBLF 算法.

```
1.assign the average computation data to every task;
2.compute the tlevel and blevel of all tasks;
3.while not all tasks scheduled do
4.    select the ready nodes n_x with maximum blevel(n_x);
5.    if task n_x is a hardware task
6.                assign task n_x to hardware, schedule(n_k,hardware)=CFT
7.    else
8.          for each processor p_k in the processor-set Q do
9.                insert the task n_x into processor p_k
10.         assign task n_x to the processor p_j with minimum EFT(n_x,p_j), if there exist more
              than one processor, choose the processor with least utilization
11.   endif
12.   mark n_x scheduled, adjust the ready tasks set.
13.   compute the tlevel and blevel of all tasks according to the schedule(n_x,p_j);
14.endwhile with not all tasks scheduled
```

DBLF 算法属性:时间复杂度为 $O(n^3)$.

证明:算法至少遍历所有任务一次:总次数为 n;对于每一个任务的调度来说,将一个任务插入(insert)到处理器中的时间复杂度为 $O(n)$.为选择最佳方案,任务及其关键后继需插入到 q 个处理器中,所以插入分配处理器的时间复杂度为 $O(2n\times q)$.而且,每个任务调度后,需遍历任务图以计算动态的 *BLevel* 值,存储结构采用邻接表的图,遍历算法的时间复杂度为 $O(m)$,m 为图中边的数目.

DBLF 算法的时间复杂度为 $O(n(2n\times q+m))$,其中 m 的最大值为 n^2,且一般情况下,任务数 n 大于处理器数目 q,最后,整个算法的时间复杂度是 $O(n^3)$.而 DBLF 算法属于静态算法,是一种运行前调度(pre-run-time scheduling)算法,也就是说算法的执行时间不会影响目标系统运行时的实时性.所以,执行时间为 $O(n^3)$是可行的.

用 DBLF 算法调度如图 1 所示的任务图,设定系统的目标体系结构以如图 2(a)所示的方式互联.算法运行时任务选择顺序为{1,3,4,2,5,6,9,7,8,10},不考虑通信调度时的调度长度为 76,而文献[9]中的 HEFT 算法的调度长度为 80,CPOP 算法的调度长度为 86,用 MH 算法[7]的调度长度则为 91.在考虑通信拓扑和通信调度后,DBLF 算法的调度长度为 85,结果如图 3 所示;而 HEFT 的长度则为 86.

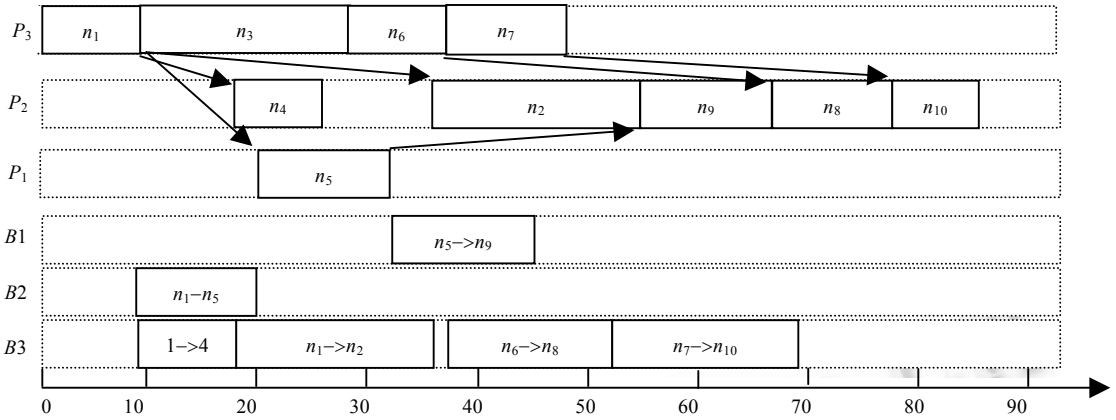


Fig.3 The schedule of the Fig.1 taskgraph generated by DBLF algorithm

图 3 DBLF 算法对图 1 的调度结果

3 实 验

为了评估 DBLF 算法的性能,我们进行了一系列的模拟研究.

调度长度在实时系统的综合中很重要,如果一个调度算法能生成更短的调度长度,就意味着当任务截止期缩短时,这个算法可以提供更高的调度成功率,也意味着可以提供更加高效的调度,因为在任务截止期与任务调度结束之间存在着空闲时间,可以作为系数调整实际运行时间和估算时间之间的不一致,使得实际调度成功率提高.因此,模拟研究中着重比较调度结果的调度长度.

3.1 比较标准

虽然调度长度是实时系统最重要的性能评价标准,但是由于在模拟中采用了大量不同性质的图形,图的调度长度会随着任务数的增加而在很大范围内波动.为了对不同算法的性能进行有效的量化比较,引入以下比较标准:

相对调度长度(normalized scheduling length,简称 NSL)定义如下:

$$NSL=\frac{schedule\_length}{\min_{p_j\in Q}\{\sum_{n_i\in CP}W_{i,j}\}}\geq 1.$$

其中,关键路径 CP 是在任务图调度前,任务执行时间等于初始值(由公式(1)决定)时的关键路径.设关键路径上的任务都分配到执行时间最短的处理器上,计算关键路径上所有任务的执行时间总和,这就是上述公式中分母值的计算结果,而这个值是给定任务图的调度长度的下限,所以无论采用何种调度算法,*NSL* 必然大于等于 1.

平均调度因子(speedup).计算串行执行时间总和,设任务图中的所有任务都分配到一个处理器上的方式是串行执行,这时任务图中所有的任务间通信时间均为 0,任务图的串行执行时间总和等于所有任务在该处理器上的执行时间总和.最大串行执行时间总和等于在所有处理器上的串行执行时间和的最大值.最大串行执行时间总和是调度长度的最大值,且调度长度可能会随着处理器个数的增加而降低,故平均调度因子定义如下:

$$speedup=\frac{\max_{p_j\in Q}\{\sum_{n_i\in V}W_{i,j}\}}{schedule\_length\times processors\_number}.$$

调度长度比较数目.在模拟试验中,用 DBLF 算法和其他调度算法对相同随机图进行调度,对生成的调度长度进行相互比较,并统计与其他调度算法相比,DBLF 算法的调度长度更短、更长、相等的任务图数目,同时将所有任务图的调度长度进行累加并相互比较结果.

3.2 随机生成图

首先考虑随机生成应用图,采用一个随机图生成器,根据不同参数生成各种特性不同的加权 DAGs.实验中

的参数如下:任务数 n、任务的最大输出度(out_degree 决定了任务图的形状和边数 m)、处理器数量 q,任务图中平均通信时间和平均执行时间之比 CCR(communication-computation-ratio).下面给出了实验中对上述参数的取值集合:

SET_N={50,100,200,300,400,500};

SET_{CCR}={0.1,1.0,10.0};

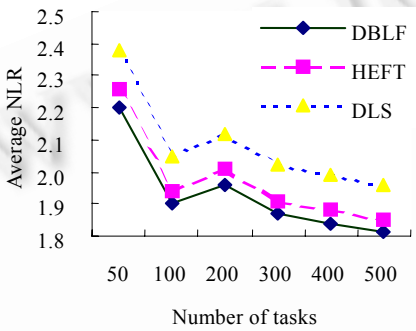
$SET_{\text{out\_degree}}$={1,3,5,n};

SET_q={4,16}.

模拟结果:上述参数可以组合成 144 种不同类型的 DAG 图,在每种类型中生成 1 000 幅随机图,所以实验中随机图生成总数大约为 144K.

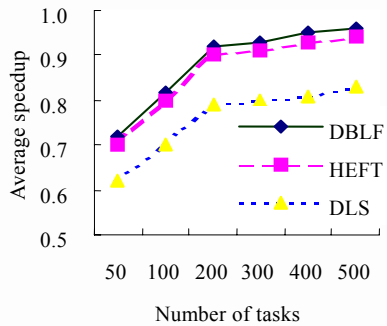
3.3 随机生成图比较结果

因为 HEFT 算法有较高的效率[9],所以选取 HEFT 算法[9]、DLS 算法[6]与 DBLF 算法进行性能比较.第 1 组实验是随着任务数增加的相对调度长度比较图(如图 4(a)所示),DBLF 算法的相对调度长度值优于 HEFT 算法 2 个百分点,优于 DLS 算法 9 个百分点.第 2 组实验是比较算法的加速比图(如图 4(b)所示),DBLF 算法也具有明显优势.这两组实验证明,DBLF 算法在降低调度长度方面有明显优势.



(a) Average NLR for different number of tasks

(a) 不同任务数的平均 NLR



(b) Speedup comparison for different number of tasks

(b) 不同任务数的平均加速比

Fig.4

图 4

另一组实验研究任务图中不同的 CCR 以及不同的处理器数目对调度质量的影响,因为 HEFT 算法总的来说优于 DLS 算法,也为了图表简洁,仅在表 1 中列出 DBLF 算法和 HEFT 算法的调度长度比较.表 1 中的“DBLF less”列表示对同一个随机任务图的调度结果,DBLF 算法的调度长度更短的统计数目;“HEFT less”列表示 HEFT 给出更短的调度长度的统计数目;“Improve”列的数据是 DBLF 算法比 HEFT 算法的调度长度降低的百分比.从表 1 可以看出,CCR 越小,DBLF 算法的改进越大;同时,处理器个数越多,改进的效果越明显.而且,随着任务数的增加,任务图的复杂度增加,DBLF 算法的调度长度比 HEFT 算法的调度长度长的可能大大降低.这是因为随着图的复杂度的增加,图中可能存在多条关键路径,而且关键路径之间互相影响,静态的 rank_u 与实际调度过程中的实际值差异可能很大,造成了 DBLF 算法的改进空间.

在模拟实验中的所有 144K 随机任务图中,调度长度小于 HEFT 算法的有 62 904 个,大于的有 2 272 个,等于的有 8 624 个,总的调度长度改进百分比为 2.58%.

Table 1 Comparison of schedule results between DBLF and HEFT algorithm
(Taskgraph No. in every row is 1000)

表 1 DBLF 算法与 HEFT 算法的性能比较(每一行的总任务图数目为 1000)

# of tasks	CCR	# of processors	Schedule length		
			DBLF less	HEFT less	Improve (%)
100	0.1	4	834	18	2.98
	1	4	852	7	2.75
	10	4	705	81	1.56
	0.1	16	858	3	9.81
	1	16	791	5	7.04
	10	16	646	103	2.06
500	0.1	4	917	0	2.52
	1	4	887	2	2.35
	10	4	843	1	2.02
	0.1	16	924	0	2.59
	1	16	893	0	3.39
	10	16	857	0	2.22

4 结 语

调度算法是异构分布式系统的性能目标能否实现的关键因素,由于调度算法的重要性,一直是分布式系统中的研究热点.大多数的调度算法集中在同构系统上,近来也有不少异构系统的启发式调度算法,但是这些调度算法大多不考虑系统网络的互连拓扑结构和通信对调度的影响,而通信对于实时嵌入式系统的时间性能分析至关重要.本文在研究了近几年的调度算法之后,提出了一种将应用任务图调度到异构嵌入式系统中的 DBLF 算法,根据模拟研究可以看出,新算法在调度长度方面明显优于已有的算法,如 DLS 算法、HEFT 算法和 CPOP 算法等,可以有效地降低任务图的调度长度.

References:

[1] Yen TY, Wolf W. Hardware/Software Co-Synthesis of Distributed Embedded System. Netherlands: Kluwer Academic Publishers, 1996. 1~57.

[2] Garey MR, Johnson DS. Computers and Intractability—A Guide to the Theory of NP-Completeness. New York: W.H. Freeman and Co., 1979.

[3] Kwok YK, Ahmad I. Dynamic critical-path scheduling: An effective technique for allocation task graphs to multiprocessors. IEEE Trans. on Parallel and Distributed Systems, 1996,7(5):506~521.

[4] Hou ESH, Ansari N, Ren H. A genetic algorithm for multiprocessor scheduling. IEEE Trans. on Parallel and Distributed Systems, 1994,5(2):113~120.

[5] Wu M, Gajski D. Hypertool: A programming aid for message passing systems. IEEE Trans. on Parallel and Distributed Systems, 1990,1(3):330~343.

[6] Sih GC, Lee EA. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans. on Parallel and Distributed Systems, 1993,4(2):175~186.

[7] EI-Rewini H, Lewis TG. Scheduling parallel program tasks onto arbitrary target machines. Journal of Parallel and Distributed Computing, 1990,9(2):138~153.

[8] Wu MY, Shu W, Gu J. Efficient local search for DAG scheduling. IEEE Trans. on Parallel and Distributed Systems, 2001,12(6): 617~627.

[9] Topcuoglu H, Hariri S, Wu MY. Performance-Effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. on Parallel and Distributed Systems, 2002,13(3):260~274.