函数型程序的并行计算模型及任务划分*

周一苇[1] 郑守淇[2] 白英彩[1]

[1](上海交通大学计算机科学系 上海 200030)

[2](西安交通大学计算机科学系 西安 710049)

E-mail: ypzhou@csg.sjtu.edu.cn

摘要 通过分析得出了函数型程序的并行计算模型——任务树,并应用该模型分析了任务划分中的任务粒度和并行度等主要因素对加速比的影响,提出了优化的任务划分算法,最后给出了在一个分布符号处理系统PARLisp中的实现结果.

关键词 函数型程序,并行,分布式计算,任务划分.

中图法分类号 TP311

随着硬件成本和通信费用的下降,将多台计算机以通信介质相连使资源共享的方式,尤其是共享处理机资源的计算机集群等方式日益受到关注,并行计算迅猛发展.传统的并行计算模型是任务图,基于任务图的并行计算在分布式数值处理中取得了很大成功,如,大规模并行处理机MPP(massively parallel processor)上的天气预测计算.但是,函数型语言与过程型语言不同,具有无副作用、数据结构不规则等特点,任务图模型是否也适合于面向分布式符号处理的并行函数型程序呢?

任务划分是指,将应用程序分割成若干并行计算的单位.任务划分受程序结构的制约,其优劣影响到程序的并行执行效率.在应用程序的划分中存在的典型问题是:并行任务是否越小、越多,就越好;并行化的次数越多,程序执行是否就越快.

本文首先讨论并行函数型程序的计算模型,在此基础上分析任务划分与程序加速比的关系,提出优化的划分算法,最后给出一个并行函数型语言系统的实测数据.本文的讨论面向分布存储和消息传递平台上的并行.

1 计算模型

1.1 任务图

常用的并行计算模型是任务图.程序 p 的任务图是一个有向无环图 $G_p=(V,E)$,$V=\{t_1,t_2,\ldots,t_n\}$是由 p 划分成的任务的集合.[1]每个任务只能在单个处理单元上执行,执行中不和其他任务通信或同步.$E\subseteq V\times V$,$(t_i,t_j)\in E$ 当且仅当由于 p 的数据依赖关系或任务可步要求 t_i 在 t_j 前执行,由于 G 无环,E 为任务集 V 上的偏序.

任务图在并行科学计算中得到了广泛应用,如向量机上的并行Fortran程序,但其应用多为在共享存储平台上的中小粒度并行.

1.2 任务树

不同于传统程序,函数型程序的数据处理由函数调用完成.例如,对Lisp语句$(F\ arg_1\ldots\ arg_n)$,先求 n 个输入参数的值,再执行函数 F,得到结果.在传统程序中,对数据进行写操作的指令为引用此值的指令提供信息,因此,前者必须在后者之前执行,由程序计数器使相关的指令按序执行.因此,过程型程序的代码段自然地对应到任务图中的结点.而在函数型程序中,上面所述的数据间的输入/输出关系表现在函数间,每个函数将其他函数的输出数据作为输入参数,因此执行次序依赖于函数.[2]形如$(F(F_1\ldots)\ldots(F_n\ldots))$的嵌套调用可表示为一棵函数调用树,树的节点为函数,树枝为调用关系,树表示出了数据的流动.兄弟节点间无数据依赖,它们的执行顺序与执行结果无关.

---

* 本文研究得到国际合作项目基金资助.作者周一苇,女,1973年生,博士生,主要研究领域为分布式智能化计算,计算机网络管理.郑守淇,1927年生,教授,博导,主要研究领域为计算机系统结构,分布式智能化计算及应用.白英彩,1936年生,教授,博导,主要研究领域为分布式系统,通信网络.

本文通讯联系人:周一苇,上海200030,上海交通大学计算机科学系

本文1997-07-25收到原稿,1997-12-18收到修改稿

并行函数型程序执行中较典型的一种情况是,任务在执行过程中动态产生若干子任务.子任务并行执行,直到所有子任务结束后父任务才继续执行.因此,并行函数型程序的基本模型为树$\langle Tsk, Tr\rangle$,Tsk 是任务集$\{t_0, t_1, \ldots, t_n\}$,$Tr$ 是 Tsk 上的偏序,t_j 是 t_i 的儿子,当且仅当根据数据依赖关系,t_j 在 t_i 产生任务后开始,回收结果前结束.叶节点是串行任务,其余节点是并行任务.

以一个进行列表元素排序的 Lisp 程序为例,程序结构如下所示:

```
1  (Function sort (list)
2    (if list is empty
3        then (return list;)如果为空表,则直接返回.
4        else ((smllist,lrglist)←(partion list);拆分成两个子表,smllist 表中的元素比 lrglist 的元素小.
5          lrglist←(sort lrglist); smllist←(sort smllist);分别对子表排序.
6            list←(merge smllist lrglist);合并排序后的子表.
7            return list;)))
```

该程序的任务树如图 1 所示.

如果采用任务图表示,则树中的结点需分成若干小结点,如图 2 所示.图中(sort ...)结点又可细化为相似的任务图,因此,采用任务图后,任务的粒度减少了,任务间的关系也复杂了,并行化的开销必将急剧增大,因而,在基于消息传递的系统中性能难以提高.

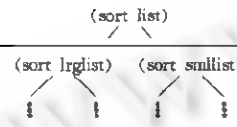

图1

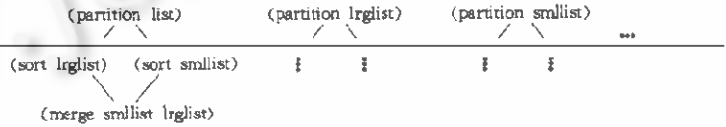

图2

综上所述,任务树具有以下特点:

(1) 任务树比任务图结构简单,更易于分析、实现;

(2) 任务树更接近于函数型程序的语义,程序更易于对应到任务,划分的实现简单;

(3) 任务粒度偏大,适合于分布存储和消息传递系统上的并行.

在以下的讨论中假设:

(1) 任一时刻任一处理单元上只运行一个串行任务,或运行一个并行任务及其某一子任务;

(2) 一个任务只能在一个处理单元上执行,执行中只与其子任务通信、同步;

(3) 任务调度固定不变.

设任务 t_i 的开始时刻为 S_i,其有效计算时间,亦即串行执行时间为 T_i,实际计算时间,亦即从任务开始到结束经过的时间为 R_i,任务集合中根任务为 t_{root},那么,

(1) 若 t_i 为串行任务,则 $R_i=T_i$;

(2) 若 t_i 为并行任务,则 $R_i=T_i+\max\limits_{t_j\text{的儿子}}(S_j-R_j)-\min\limits_{t_j\text{的儿子}}S_j+T_c$,其中 T_c 为并行开销;

(3) 程序的串行运行时间 $T_{SEQ}=\sum T_i$;

(4) 程序的并行运行时间 $T_{PAR}=R_{root}$.

2 任务划分

2.1 任务划分与加速比

加速比是衡量程序并行执行效果的最常用的指标之一,是程序串行与并行执行时间之比.如果并行处理单元数为 n,那么加速比 $1\leqslant S\leqslant n$.如果处理单元足够多,任务调度算法能把并行任务分派到不同的处理单元上执行,因而任务划分直接影响加速比的大小;否则,任务划分与任务调度策略的优劣共同影响加速比.

假设:(1) 处理单元足够多,需要并行的任务都能在不同处理单元上同时执行.

(2) 考虑到同一任务的子任务一般能接近同时开始执行,忽略兄弟任务间开始时刻的差异.

不同的任务划分法对应于不同结构的任务树.只有两层的任务树如图 3 所示,即子任务内禁止嵌套并行.

设并行的额外开销为常量 T_c,则加速比

$$S=\frac{T_0+T_1+\ldots+T_m}{T_0+\max(T_1,\ldots T_m)+T_c}.$$

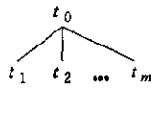

图3

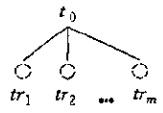

图4

如果 T_0 和 $T_1+\ldots+T_m$ 固定，则当 $T_1=\ldots=T_m$ 时，S 达到最大.

结论 1. 禁止任务嵌套并行时，如果根任务大小不变，那么各子任务大小越相近，加速比越大.

如果 $T_1,\ldots,T_m$ 固定，则 T_0 越小，S 越大. 特别地，在 $T_1=\ldots=T_m$ 时，$S=\dfrac{m+T_0/T_1}{1+T_0/T_1+T_c/T_1}$ 时，当 $(T_0+T_c)/T_1\to 0$ 时，$S\to m$.

结论 2. 禁止任务嵌套并行时，如果子任务大小不变，那么根任务越小，加速比越大.

如果允许子任务嵌套并行，则 $t_1,\ldots,t_m$ 处是子任务树 $tr_1,\ldots,tr_m$，如图 4 所示.

任务树中存在一条关键路径 $(t_{i0},t_{i1},\ldots,t_{iN})$，$t_{i0}$ 为根节点 t_0，$R_{ik}=\max_{t_{ik-1}\text{的儿子}} R_j$，$R_{in}=T_{in}$，程序 p 的并行执行时间 $T_{PAR}=\sum_0^n T_{ik}+nT_c$.

结论 3. 允许任务嵌套并行时，程序的并行开销随着关键路径长度线性增加，与任务树的深度(即并行的最大嵌套层数)无直接关系.

为方便讨论，令 $T_C=0$. 设 $tr_1\ldots tr_m$ 的串行执行时间为 $T_1\ldots T_m$，并行化加速比为 $S_1\ldots S_m$，则

$$S=\frac{T_0+T_1+\ldots+T_m}{T_0+\max(T_1/S_1,\ldots,T_m/S_m)}.$$

设 $T_1=\ldots=T_m$，$T_0/T_1=a$，则

$$S=\frac{m+a}{1/\min(S_1\ldots S_m)+a}.$$

因此，结论 2 在嵌套并行情况下仍成立. 由于 $\min(S_1\ldots S_m)\geqslant 1$，可以得到结论 4.

结论 4. $T_1=\ldots=T_m$ 时，加速比的下限为 $\dfrac{m+a}{1+a}$.

2.2 任务划分算法

根据 1.2 节的任务树模型和 2.1 节中基于此模型的分析，合理的任务划分策略可以控制任务大小及嵌套并行等因素，减少并行开销，使加速比达到最大. 我们提出了如下的划分算法，并给出必要的任务调度方法：

(1) 首先进行任务并行化，即将一个串行任务转换为语义相同的一棵任务树. 采用广度优先的递归派生子任务的算法：

(1.1) 对串行任务 t_0 进行语义分析，划分出其中可并行执行的部分(如对一个列表的各元素逐个进行相同操作、各次循环无依赖关系的迭代等)，将其尽可能多地分割成若干相互独立的并行子任务 $t_1\ldots t_m$.

说明：并行部分计算量越大，子任务数越多，并行效果越好.

(1.2) 假设父任务 t_0 和子任务 $t_1\ldots t_m$ 的有效执行时间为 $T_0\ldots T_m$，且 $T_1\ldots T_m$ 由大到小排列. 若 $(T_1+\ldots+T_m)/T_0<e$，则放弃对 t_0 的并行化；否则，就将 t_0 替换为由父结点 t_0 和子结点 $t_1\ldots t_m$ 组成的两层树；

说明：e 为获得一定加速比所需的最小并行部分比例，可通过运行简单的基准程序测得. 如果并行部分过小，并行开销会抵消或超过并行收益，不必并行执行.

(1.3) 如果 t_0 有子任务，则逐个对子任务 $t_1\ldots t_m$ 进行(1.1)和(1.2)的并行化操作. 如果存在 $t_i(0<i<m)$ 不能并行化，则放弃对 $t_{i+1}\ldots t_m$ 的并行化；否则，依次将 $t_1\ldots t_m$ 替换为两层的树. 再将任务树中每层的最左边结点的执行优先权置为最高.

说明：如果执行时间较长的子任务无法再提高速度，则其兄弟任务必定不在关键路径上，也就不必进行并行化；而对于关键路径上的任务，应得到最多的机会以提高速度.

(1.4) 如果 $t_1\ldots t_m$ 有子任务，则对子任务进行(1.3)的操作. 依此类推，对任务树中最高层的结点进行(1.3)的操作，逐层伸展任务树，直到任务树不能再伸展.

(2) 再将任务树中过小的串行任务合并. 设任务树最高层的结点 $t_{i1}\ldots t_{in}$ 是 t_{i0} 的子任务，且 $T_{in}+\ldots+T_{ij}<T_{i1}$，$T_{in}+\ldots+T_{ij}+T_{ij-1}>T_{i1}$，则将 $t_{in},t_{in-1},\ldots,t_{ij}$ 合并为一个任务.

说明：合并是为了使这些任务在同一处理单元上执行，在处理单元有限的情况下，可增加其他任务获得处理单元的机会. 这种方法减少了任务分配次数，比在任务分配阶段将其分配到同一处理单元上效率要高.

该任务划分算法有如下特点：

* 保守型的并行. 通过控制任务粒度和嵌套并行，不并行效果不好的串行任务，使效果较好的任务有更多的机会

获得处理单元，并尽量减少并行开销．与保守型相反的是贪婪型．在处理单元争用激烈和并行开销较大的场合，保守型并行能够保证一定的加速比．

* 带有优先权的并行．分配到同一处理单元上的任务中，优先权高的先执行．优先权为提高程序加速比提供了并行控制的有力手段．

3 实　例

PARLisp 是由西安交通大学新型机研究室实现的一个并行 Lisp 语言解释系统，它为目前的基于消息传递和分布存储的新型计算机体系结构(如 Cluster，MPP)提供了一个理想的分布符号处理平台．PARLisp 系统的实现基于通用平台和抽象的消息传递模型，支持分布存储系统上中大粒度显式并行，对并行 Lisp 程序解释执行．

3.1 任务树模型和任务划分算法的应用

该系统以任务树为模型，由此模型确定了系统的并行机制：

* 并行单位．在任务树模型中，较大的并行粒度才能保证并行开销足够小．因此，进行任务级的并行，而不是指令级的并行；并行任务对应于 Lisp 函数调用的一个序列，并允许其中的函数调用并行执行；通过提供相应的语言机构，根据 2.1 节的结论，由系统及应用程序共同实现 2.2 节中提出的任务划分算法，控制并行的嵌套和任务的大小，应用程序还可实现并行优先权控制．

* 控制机制．由任务树的定义，父任务产生子任务，子任务又可产生子任务，但子任务只有一个父任务，一个或多个子任务合作为，父任务服务．

* 数据处理机制．由任务树的定义，父子任务、兄弟任务间无数据共享．父任务产生子任务时，可以设置子任务的初始求值环境(即上下文)，任务开始执行后，只访问自己的求值环境．

* 通信．由任务树定义中父子任务间的依赖关系，父任务产生子任务时，向子任务传递参数，并设置求值环境，任务开始执行后无通信，直到任务结束，向父任务返回结果．

* 同步．由任务树定义中任务间的依赖关系，任务间无共享数据，不存在数据存取同步，但存在并行任务的执行顺序同步，这有两种情形：

保守并行：父任务等待所有的子任务都返回结果后继续执行；

投机并行：父任务只等待某个子任务返回结果后就可继续执行．[8]

3.2 系统性能

PARLisp 对于一组常规的基准程序获得了较好的加速比，测试结果如表 1 所示．其中，系统的测试环境为：

(1) 曙光 1000 大规模并行计算机系统．32 个基于 i860 的主频 40MHZ 的计算节点用 2D-MESH 通信网连接，网络总通信容量为 4.8GB/s，节点机与网络通信总带宽为 2.8GB/s；

(2) 8 台 SUN SPARC 20 工作站，用 10BASE5-ETHERNET 相连，网络通信速率为 100Mbps，高效精简网络协议以及硬件支持，包传输时间<1ms；

(3) 4 台 P5/75 以 10BASE5-ETHERNET 形式相连．[9]

表 1

基准程序	测试环境	平均执行时间(s)	平均加速比
PBOYER	曙光 1000，1 个节点	6.20	1.00
	曙光 1000，4 个节点	2.03	3.05
	曙光 1000，8 个节点	1.25	4.96
PBYR3	(SPARC 20)×1	2.84	1.00
	(SPARC 20)×2	1.52	1.87
	(SPARC 20)×4	0.74	3.84
	(SPARC 20)×6	0.69	4.12
PTAK1	(P5/75)×1	5.77	1.00
	(P5/75)×4	1.80	3.20
PTAK2	(P5/75)×1	5.77	1.00
	(P5/75)×4	5.39	1.07

测试的基准程序和在程序中采用的任务划分策略如下所示：

(1) PBOYER．该程序进行定理的机器证明．由 GABRIEL 的 BENCHMARK BOYER 并行化而成，对其中的

REWRITE—— ARG 部分并行化. 在串行 BOYER 中,逐个对表中的各元素进行 REWRITE 函数调用,由于 REWRITE 函数调用的复杂度随着表的长度和深度的增加而增加,但组成表的每个元素的长度和深度无法预测,只能禁止计算量小的 REWRITE 函数调用并行执行,因而将串性程序改成:若表中的元素为长度超过 5 的表,则 REWRITE 函数调用并行执行,否则串行执行;

(2) PBYR3. PBOYER 的另一版本,计算量比 PBOYER 小,但采用相同的任务划分策略;

(3) PTAK1. 由 GABRIEL 的 BENCHMARK TAK 并行化而成. 在串行 TAK 中,TAK 函数 4 次递归调用自己,前 3 次的结果作为第 4 次调用的参数,将前 3 次的函数调用改为由不同的处理单元并行调用 TAK. 在程序中采用 2.2 节中的算法控制并行嵌套;

(4) PTAK2. 该程序功能同上,但采用贪婪式并行,将每个可并行的任务并行化.

关于 Lisp 等函数型程序的并行化,国内还未见到类似的成果,而国外并行 Lisp 的研究可分为两大类:① 针对特定的多处理机系统研制的,如,Cmlisp 是为 65 536 个处理器 SIMD 的并行机 Connection Machine 设计的,Multilisp,Qlisp,Spur Lisp,Balinda Lisp[2,4,5]是基于共享存储的细粒度、大规模并行的,这与基于任务树模型的系统完全不同,像 PARLisp 这样适用于基于消息传递的分布存储系统的并行 Lisp 还很少见;② 针对一般的集群系统,但多为编译型系统,而基于任务树模型的系统可灵活地采取编译或解释的执行方式,同时,优化的任务划分算法为系统性能提供了保证,PARLisp 现在采用了解释执行方式,稍作修改即可采用编译方式.

4 小 结

因此,任务树是适合于函数型程序中大粒度并行的一种计算模型,它实现简单而灵活. 本文在此基础上分析了任务划分策略和加速比的关系,提出了优化的划分算法,一个实现了的并型函数式语言系统 PARLisp 的实测数据证明了任务树模型及任务划分算法是正确高效的. 另一方面,任务树模型也限制了并行任务间的交互,而且在实际系统中,任务量预测以及处理单元互连的复杂性影响了任务划分的优化算法取得理想的效果,大大增加了任务划分算法实现的难度,如何解决这些问题值得作进一步的研究.

参考文献

1 Donaldson Val, Berman Francine, Paturi Ramamohan. Program speedup in a heterogeneous computing network. Journal of Parallel and Distributed Computing, 1994,21(3):316~312

2 Yuen C K, Feng M D, Wang W F *et al*. Parallel Lisp Systems. Singapore: Chapman and Hall, 1993

3 周一萍. 基于分布存储及消息传递系统的分布式智能软件开发环境——PARLISP 的设计与实现[硕士论文]. 西安交通大学, 1996

(Zhou Yi-ping. Design and implementation of PARLISP—— a distributed AI programming environment based on distributed memory and message passing platform[M.S. Thesis]. Xi'an Jiaotong University, 1996)

4 Halstead Robert H. Multilisp: a language for concurrent symbolic computation. ACM Transactions on Programming Languages and Systems, 1985,7(4):501~538

5 Almasi G S, Gottlieb A. Highly Parallel Computing. Redwood City, CA: Benjamin/Cummingsm Publishing, 1989

Parallel Functional Program Computing Model and Task Partition

ZHOU Yi-ping[1] ZHENG Shou-qi[2] BAI Ying-cai[1]

[1](*Department of Computer Science Shanghai Jiaotong University Shanghai* 200030)

[2](*Department of Computer Science Xi'an Jiaotong University Xi'an* 710049)

Abstract In this paper, task tree is introduced to model a parallel functional program on distributed and message passing platform. Based on this model, the main issues of task partition, such as task size and parallelism, are analyzed with speedup as the performance measure. An optimized partition algorithm is given, and an implemented system——PARLisp's performance data is supplied as an example.

Key words Functional program, parallel, distributed computing, task partition.