并行任务动态派生的积极惰性化控制方法*

田新民　王鼎兴　沈美明　郑纬民

(清华大学计算机科学与技术系,北京 100084)

摘要 本文给出了并行任务派生的理想状态,分析和研究了积极任务派生(ETD)方法和惰性任务派生(LTD)方法,指出了这两种方法所具有的局限性,提出了一种新的并行任务派生的积极惰性化方法(ELDT)及其算法.初步研究表明ELDT方法可安全有效地增大计算粒度,在由多个商售单处理器构成的小规模并行系统上ELDT算法有效地控制计算粒度和任务派生,使并行任务的派生近似达到理想状态.

关键词 并行任务,动态派生,任务粒度,惰性任务派生.

并行任务的动态派生是并行多机系统调度和管理并行计算任务的关键问题.任务派生的合理与否在一定程度上决定了任务粒度(task granularity)是否与并行系统的资源规模及能力匹配.目前并行任务动态派生的控制方法主要有两种.一种是积极的任务派生方法(Eager Task Deriving),简称为ETD方法.该方法根据程序中的任务划分将尽可能积极地派生计算任务,而不考虑处理机资源的变化.另一种并行任务派生方法是惰性派生方法(Lazy Task Deriving),简称为LTD方法.其中Lazy的含义是只有存在可利用的处理机资源时,才补偿性(retroactivity)地派生计算任务[5].由于许多并行算法的计算粒度通常细于MIMD系统能有效开发的并行计算粒度,因此许多研究人员试图通过设计专用硬件支持细粒度计算[3],但是这种方法受限于任务间的数据相关性使硬件价格过于昂贵,故而有很大的局限性.合理的粗粒度任务派生需要对计算开销进行估计[4,2],而且计算任务的输入数据集是动态变化的,编译时静态估计计算开销极为困难,因此运行时(run-time)动态控制并行任务的派生就成为一个很有意义的研究课题.本文给出了并行任务派生的理想状态,并据此分析和研究了ETD和LTD方法,指出了这两种方法的局限性,进而提出了一种并行任务派生的积极惰性化方法及其算法,我们称之为ELDT(Eager-Lazy Deriving Tasks)方法.初步研究表明ELDT方法可安全有效地增大并行计算的粒度,ELTD算法能使并行任务的派生近似达到理想状态.

* 本文1991年6月1日收到,1991年11月21日定稿

本课题受国家863高技术项目863-306-101和国家高校博士学科点专项基金0249136资助.作者田新民,30岁,博士生,主要研究领域为并行/分布计算机系统.王鼎兴,57岁,教授,主要研究领域为并行处理与智能计算机系统.沈美明,女,56岁,副教授,主要研究领域为计算机系统结构.郑纬民,48岁,教授,主要研究领域为并行/分布计算机系统.

本文通讯联系人:田新民,北京 100084,清华大学计算机系

© 中国科学院软件研究所 http://www.jos.org.cn

1　并行任务派生的理想状态

许多并行算法所描述的计算粒度为细粒度,对应的计算任务派生后用计算树表示.例如,并行算法 Divide—And—Conquer(DAC)对应的计算树是一棵完全二叉树.DAC 算法描述如下:

Step1:确定计算任务的根结点

Step2:根结点将任务划分成两个子任务,并将它们分派给左右子树的根结点

Step3:以递归方式重复 Step2,直到计算任务的分解满足递归的终止条件

Step4:叶子结点进行计算,并将计算结果传给它们的双亲结点

Step5:双亲结点接受左右子树根结点的计算结果进行计算,并将计算结果向上层双亲结点传送

Step6:以递归方式重复 Step5,直到根结点的任务计算完毕

图 1 给出了 DAC 算法排序[6 4 5 7 3 11 21 28 1 2 34 8 9 18 14 67]的计算树,其中每个结点的计算任务是派生子任务,等待左右子树的计算结果进行归并排序并将排序结果传向双亲结点.如根结点(root)派生的两个子任务是 T1,T2;T1 排序[6 4 5 7 3 11 21 28],T2 排序[1 2 34 8 9 18 14 67],T0 归并排序 T1 与 T2 的排序结果.计算树中{T7,T8,…,T14}为叶子结点,被视为最简计算任务,不再具有计算并行性.

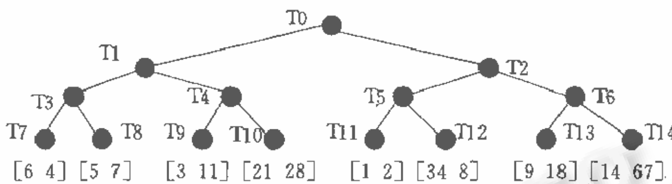

图1　DAC算法排序树

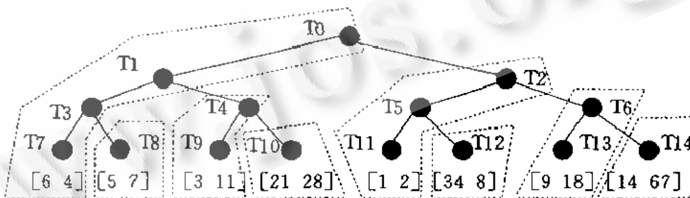

图2　8个PE的任务分派

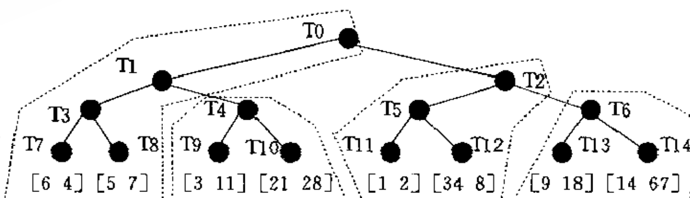

图3　4个PE的任务分派

考察图 1 可以发现,随着排序表长度的增加,并行任务数也将线性增长.设表长为 L,则叶子结点数=[L/2],DAC 算法派生的总任务数=L—1,此时若每个计算任务都占用一个

© 中国科学院软件研究所　http://www.jos.org.cn

处理器,那么将需要 L−1 个处理机,即所需要的处理器将随任务数线性增长,然而资源限定的并行系统显然不能满足这种要求;另外细粒度任务(如:DAC 算法描述的任务粒度)将会引起的大量的同步通迅延迟,任务派生和任务切换开销,这将大幅度降低系统性能.因此理想状态的并行任务派生是在保持负载平衡的情况下,并行任务具有与系统资源相匹配的最大运行时(run−time)粒度.为此在任务派生任务时应首先采用宽度优先(breadth−first)方式展开计算树直到所有处理机处于繁忙状态,宽度展开则转化为深度优先(depth−first)的计算方式.图 2、图 3 分别给出了上例在具有 8 个 PE,4 个 PE 的并行系统中理想的任务分派状态.

为了获得有效的任务分派,采用静态技术,如程序变换技术增大计算粒度[1].但是单纯的静态方法由于缺乏运行时处理器资源变化的信息而难以获得满意的并行任务派生.为此必须考虑采用动态的并行任务派生方法.

2 并行任务派生的动态控制

并行任务派生的动态方法一般可分为两种,即积极任务派生(Eager Task Deriving, ETD)和惰性任务派生(Lazy Task Deriving, LTD).采用 ETD 方法意味着直接根据程序中并行任务的划分派生任务,在程序执行过程中当且仅当有可派生任务时即生成一个新任务,而根本不考虑系统资源是否能有效地支持该任务的并行执行.ETD 主要侧重于开发计算并行性,其任务粒度主要取决程序对任务的划分,一般情况下 ETD 所形成的任务粒度过细,并且由于各任务间的数据相关性的影响,通常会导致大量的通讯、同步和任务生成等额外开销.ETD 的局限性可具体地概括如下:

1. 采用 ETD 方法会导致许多无效的并行计算和额外开销.

2. ETD 方法不考虑系统资源的规模和能力,所形成的任务分派与系统资源不匹配.

3. ETD 方法将可能生成远远超过理想状态的计算任务,并且任务数指数爆炸.

4. ETD 方法不能提高细粒度递归计算的效率,有时反而会降低其计算效率.

与 ETD 方法对应的惰性任务派生方法(LTD)则主要从系统资源和能力的角度出发考虑开发与系统通讯和计算能力匹配的并行性,其核心思想是当且仅当存在一个处理机空闲时,它可向有关繁忙处理机发出任务请求(request),接到请求的处理机则根据本处理机的是否有可派生任务,若有可派生任务则生成一个新任务发往请求任务的处理机.B. Kranz, E. Mohr 等人在 Mul−T 系统的研制中采用了 LTD 方法[5].LTD 方法的局限性可概括为:

5. 在计算树的不平衡性的情况下,LTD 方法对计算粒度的改变没有影响.考察例 1 可以发现,并行映射函数 map 以线性递归方式作用于表结构,如果函数 f 为细粒度函数,那么额外的任务 f 分派、迁移、同步通讯开销将会降低系统性能.

6. 纯 LTD 方法在任务窃取时机不合适时可能会导致丧失计算的有效并行性,降低资源利用率.

7. 在 LTD 方式控制下各处理器再次分派或接受任务时可能丧失计算的局部性,使并行任务进行不必要的远程迁移.

© 中国科学院软件研究所 http://www.jos.org.cn

例 1：

```
fun parmap_head f nil = nil
  |parmap_head f h::t =
    let
      val h' = f(h)
      || t' = parmap_head f t
    in
      h'::t'
end
```

```
fun parmap_tail f nil = nil
  |parmap_tail f h::t =
    let
      val t' = parmap_tail f t
      || h' = f(h)
    in
      h'::t'
end
```

为了避免 ETD 和 LTD 方法所具有的局限性，本文提出了一种新的并行任务分派的积极惰性化混合方法(Eager－Lazy Deriving Tasks method，ELDT)，ELDT 方法以积极方式派生虚拟的并行计算任务，以惰性方式控制运行时真实的任务派生.空闲处理机在窃取计算任务时采用 Oldest－first 策略，源处理器的计算在任务控制栈中的推进采用 Last in first spark 策略，这样任务的动态派生就与源处理器的计算推进行为统一起来，从而可以避免许多额外的任务切换开销(context switch overhead)，使 ELTD 算法能有效地增大运行时任务粒度，减小细粒度计算所造成的额外的运行时任务派生和计算同步开销.下节将介绍 ELTD 方法原理及其算法.

3　ELDT 方法原理与算法

ELDT 方法的基本原理是在程序的执行过程中当存在空闲处理器时，各个空闲处理器向近邻处理机发出任务请求信号，繁忙处理器根据当前的任务执行情况响应或屏蔽请求，若响应则允许其采用 Oldest－first 策略主动地窃取可发出的并行任务(stealing exportable parallel task).各处理器都带有一个任务控制栈 T，并在任务执行过程中积极地为各个可派生的本地任务在控制栈中保留充分的计算环境，栈 T 是使计算的推进与任务派生一体化的中间控制结构，这种一体化的好处在于任务的派生机制融合到了计算中，从而能减少许多为开发并行性所引起的额外开销，使开发计算并行性所获得的效益大于开发计算并行性带来的通讯，同步和任务分派等开销.各处理器上与本地任务具有同一归并点的可窃取任务称为其对应本地任务的后继(Continuation)任务，这些后继任务依据计算的顺序性存放在栈 T 中，当本地处理器执行到某任务时，若该任务尚未执行，那么本地处理器就开始计算该任务，当前这个被本地处理器执行的计算任务总是在 T 栈的栈顶.故此 ELDT 方法的优点在于能够避免纯积极(pure eager)方法派生并行任务所引起的额外开销和纯惰性(pure lazy)方法可能丧失的计算并行性.在由多个 PE 构成的 Message－passing 多机系统上，面向传统栈操作的作用式程序设计语言实现机制中，引入并行任务的派生机制时，必然涉及堆栈的分裂和局部计算环境的传输，因此必须建立简单有效的计算和控制环境[7].ELDT 所采用的控制环境及计算环境如图 4 所示.

根据图 4 所示的计算环境和任务环境组织，ELDT 算法描述采用通常的生产者－消费者模型(Producer－Consumer Model).形成惰性后继的处理机作为生产者，窃取惰性后继的处理机为消费者.任务执行过程中，生产者将可派生的本地任务压入控制栈中的惰性后继队列，由 LT_Head 和 LT_Tail 构成的后继队列向下增长，任务框架向上增长.这种控制方式使消费者在窃取惰性后继时能有效地收缩栈空间和堆空间，便于回收废片空间.

© 中国科学院软件研究所　http://www.jos.org.cn

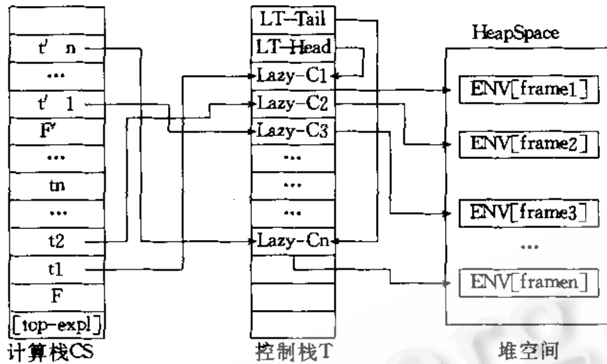

图4　ELDT的计算和控制环境

消费者在窃取本地任务时首先向生产者发出请求信号，生产者通过指针 LT_Head 确定最先形成的后继(Continuation)任务，再通过指针 Lazy_C_i 确定所对应的计算环境，然后将该环境发送给消费者. 生产者在完成任务派生后加锁计算栈中相应的 C_i 单元的锁位，直到消费者返回该后继任务的结果，再解锁 C_i 单元的锁位，并将结果填入计算栈中的 C_i 单元. ELDT 算法描述如下：

```
Procedure ELDT_Producer(T)      /* Algorithm 1 */
Begin
    if Can_partitioning(T) then
      Partitioning(T) =>[T1,T2,…,Tn]   /* [...] is a set of partitioned tasks */
      Push Tn,…T3,T2 into Controlling_stack while Ti is't atomic op
      Push Cn,…C3,C2 into Computing_stack /* Ci is the abbre. of Ci{…} */
      open interrupt, Recursive Call ELDT_Producer with T1
      if (T1 have Continuative Tasks) then
        flag <=get_result(T1's Continuation)
        if flag<>true then
          Blocking T until C2,C3,…,Cn Executing end
        endif
      endif
    endif
    Executing T in Computing--stack
End.

Procedure ELDT_Consumer      /* Algorithm 2 */
Begin
    while .T. do
      begin
        if idle(Processor) then
          Send eager request signal for Executing Tasks to
          Contiguous processors,
          Waiting for response signal and task T
          if receive(T) then
            starting ELDT_producer with task T
          endif
        endif
      end
End.
```

```
Procedure ELDT_interrupt_handle      /* Algorithm 3 */
Begin
    if (there are continuations in Controlling_stack) then
       Sending Oldest Continuation Ci pointed by LT_head to the
       idle processor that sent eager request signal.
       LT_head downward
       free frame Fi to which Ci responsed
    else close interrupt
    endif
End.
```

ELDT 算法中的 $C_K(t1,\cdots,tn)$是 T_{K-1}的后继控制,是 T_{K+1}的前趋控制,$\{t1,t2,\cdots,tn\}$为 $C_K(\cdots)$的环境.需要说明的是 ELDT 算法中的中断处理机制采用传统的中断机制,ELDT_producer 能即时地响应中断请求并转向中断处理.另外根结点的结果计算出后,根结点所在的处理机向整个并行系统广播停机信号,终止各处理器上各个进程的执行,这种方法能有效解决计算的停机问题.n一皇后问题,Fibonacci 函数的计算和 Hanoi 塔问题都是较典型的细粒度并行计算程序,图 5 给出了采用 ELDT 方式控制的 Fibonacci 函数的计算过程.Fibonacci 函数的计算程序定义如下:

fun fib(n)=if n<2 then 1 else Add(fib(Sub(n,2)),fib(Sub(n,1)))

fib 是典型的细粒度并行计算程序,其任务数随输入参量 n 的增长而指数增长,因此采用纯积极方式进行计算所引起的任务频繁创建、通讯和同步开销将抵消开发计算并行性所带来的时间效益,为此必须考虑任务派生的合理控制.图 6 给出了基于 ELDT 控制 fib 并行计算的实验结果.

假设并行系统由两个处理器 PE1,PE2 构成,fib(n)的计算如图 5 所示,其中操作子 Ce 表示后继可输出操作,Cr 是后继替代操作(即用其它处理机返回的计算结果重写栈顶的 Cr,并具有同步作用),操作子 Cb 表示可输出后继操作,Cb 将计算结果送向源处理器.VAL 表示计算结果,ENV 表示计算环境,PEi.Adr 为源处理机地址.[exp]表示计算表达式 exp.由于 ELDT 方法基于传统的栈操作,计算推进所需要的参量都能在栈顶,即使是所发出并行任务,在栈中也有相应的控制操作.对于非结构数据均无需访问堆空间(Heap),只有计算对象为结构数据时才访问堆空间,这样可以避免象 ALICE 机那样花费许多堆空间的管理开销和访堆开销,使得所固有的顺序计算在并行系统中能维持其在高效串行系统上的效率.另外并行任务在没有被其它处理机窃取之前,若计算的推进使该任务在本处理器上执行,计算方式与传统的顺序方式基本一样,仅多一次对控制栈的间址操作,这种方式能维持较大的并行计算粒度大大减少开发计算并行性所导致的任务生成,同步通信开销,从而较大地提高系统性能.ELDT 方法是面向资源任务分派方法,具有较好的增量性和稳定性,又由于 ELDT 方法采用 Oldest—first 的任务窃取策略,因此任务窃取对被窃取任务的处理机正在推进的计算影响很小,该处理机只需响应中断,发出一个后继任务,若没有后继任务则屏蔽中断,直到存在可分派任务时再开中断.我们基于本文提出的 ELDT 方法在 3 个和 5 个 Transputer (1 个 T800,2 个或 4 个 T414)构成的并行系统上的实验数据均好于在单个 Transputer,Sun3/260 上的执行速度,其中 SML 是英国 Edinburgh 大学研制的强类型函数语言,EML 是本研究组设计的一种类 SML 语言,通过两级编译技术生成 PARALLEL C 在 Transputer 上执行.实验结果表明 ELDT 方法能有效地增大计算粒度,合理地控制计算并

© 中国科学院软件研究所 http://www.jos.org.cn

行性,从而使并行任务的派生近似理想状态.

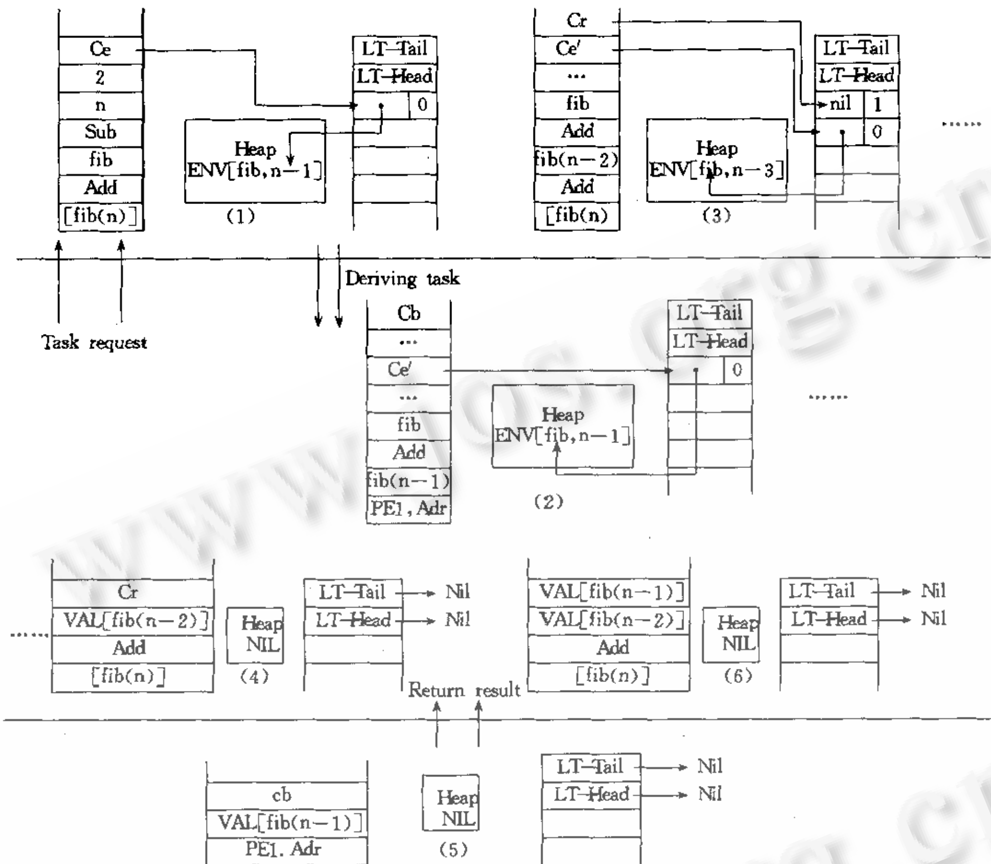

图5 Fibonacci函数的计算过程示例

n(fib's parameter)	20	22	24	26	28
SUN3/260(语言)	0.0833	0.2167	0.5333	1.3333	3.4665
SUN3/260(SML 语言)	3.4165	9.1330	23.9824	62.8808	165.1101
1 PE (1 T800)(EML 语言)	0.0955	0.2501	0.6548	1.7144	4.4883
3 PEs (1 T800 + 2 T414)	0.0493	0.1289	0.3373	0.8832	2.3121
5 PEs (1 T800 + 4 T414)	0.0305	0.0797	0.2086	0.5459	1.4291

图 6 fib 计算的实验结果(时间单位:秒)

结论:并行任务动态派生的合理控制,可以大幅度减少开发计算并行性所引起的任务生成,计算同步,通讯延迟等额外开销,使任务粒度(task granularity)与并行系统的资源规模及计算能力相匹配.本文基于并行任务派生的理想状态,提出了一种新的并行任务动态派生的积极惰性化方法(ELDT)及其算法.在多个 Transputer 构成的并行系统上的实验结果表

© 中国科学院软件研究所 http://www.jos.org.cn

明,ELDT 是一种有效的并行性控制方法,其中本文提出的积极惰性化策略,计算与控制一体化策略,并行任务窃取的 Oldest—first 策略及其计算的 Last in first spark 策略为自然地开发粗粒度计算并行性,提高由多个商售单处理器构成的并行系统的性能提供了一条有效途径. 与本文工作相关的工作,如数据相关性分析,编译时自动的计算复杂性分析,计算粒度的定量估计,资源冲突问题均为进一步的研究目标,这些工作也是目前并行处理研究领域中所涉及的基本问题.

参考文献

1 Bush V J *et al*. Transforming recursive programs for execution on parallel machines. LNCS 201 (Springer—Verlag), Sept. 1985:350—367.

2 Goldberg B. Multiprocessor execution of functional programs. Int'l J. of Parallel Programming, 1988; (**17**)**5**:425—473.

3 Gurd J, Kirkham G, Watson I. The manchester prototype dataflow computer. Communication of ACM, 1985; (**28**)**1**:34—52.

4 Hudak P, Goldberg B. Serial combinator: optimal grains of parallelism. LNCS 201 (Springer—Verlag), Sept. 1985: 382—388.

5 Kranz B, Halstead R, Mohr E. Mul—T: a high performance parallel lisp. ACM SIGPLAN'89 Conf. on Prog. Lang. Portland, OR. June 1989:81—90.

6 Milner R, Mitchell. Introduction to standard ML. Computer Science Department, University of Edinburgh, UK, 1987.

7 Tian Xinmin, Wang Dingxing, Shen Meiming. A high efficient parallel model for implementing functional languages. In: Proceedings of International Conference on Information and System, Vol. 1, Oct. 8, 1991.

A PRACTICAL EAGER—LAZY CONTROL METHOD FOR DYNAMIC DERIVATION OF PARALLEL TASKS

Tian Xinmin, Wang Dingxing, Shen Meiming and Zheng Weimin

(*Department of Computer Science and Technology*, *Tsinghua University*, *Beijing* 100084)

Abstract One of the most issues on exploiting effective parallelism is how to dynamically control parallel task derivation. In this paper, based on the ideal state of parallel tasks deriving, Eager Task Deriving (ETD) and Lazy Task Deriving (LDT) methods are analysed and discussed briefly, drawbacks of ETD and LTD are also presented. A practical Eager—Lazy method is proposed for dynamically Deriving parallel Tasks (ELDT). The exprimental results have shown that ELDT can effectively control and increase the granularity of derived tasks whose granularity is finer than the ideal granularity that multiprocessor systems can exploit efficiently, and take an ideal or satisfied state of parallel task derivation.

Key words Parallel task, dynamic derivation, task granularity, lazy task derivation.

© 中国科学院软件研究所 http://www.jos.org.cn