并行 FFT 算法在 3 种并行计算模型上的设计和分析*

陈国良　李晓峰　　黄伟民

（中国科技大学计算机系　合肥　230027）　（中国科学院合肥智能研究所　合肥　230031）

摘要　本文研究在 APRAM，BSP 和 Log$P$ 等 3 种并行计算模型上并行 FFT 算法的设计和分析；分析这 3 种模型的内在特性及其相互关系；评价它们在设计和分析并行算法时的可用性和可操作性.

关键词　并行计算模型，FFT 算法，并行处理.

当代典型的并行处理系统有多向量处理机系统（如 YMPC－90，VP2000 等）、多处理机系统（如 S－81，KSR1 等）、巨量并行处理 MPP（massively parallel processing）系统（如 Paragon，CM－5 等）和工作站网络 NOW（network of workstations）系统（如 Livermore 和 Oak Ridge 国立实验室所构造的 PVM 系统等）. 与此相应的典型的并行计算模型有基于共享存储的 SIMD 模型（如 PRAM）、基于共享存储的 MIMD 模型（如 APRAM）、基于分布存储的 MIMD 模型（如 BSP 和 Log$P$）和基于工作机群的粗粒度并行计算模型（如 C^3）.[1] 这些新近提出的更为实际的并行计算模型，均在不同程度上更加真实地反映了近代并行机的特性，从而在它们之上所设计的并行算法在实际并行机上运行时表现出较好的性能. 但是这些新模型在设计并行算法时的可操作性仍待由大量的算法实例所证实. 本文以 FFT 算法为例，来说明其在异步 PRAM（即 APRAM）模型、大同步 BSP（又叫 XPRAM）模型和分布存储 Log$P$ 模型上的设计和分析方法；阐明这些模型的演变发展过程；分析它们的内在特性和相互关系；评价它们在设计和分析并行算法时的可用性和可操作性.

本文第 1 节先简单介绍 PRAM 模型上算法设计的特点；第 2～4 节分别讨论 APRAM，BSP，Log$P$ 模型上的 FFT 算法的设计和分析方法；最后是结论.

1　PRAM 模型上算法设计特点

对于图 1 所示的 1 个 $n=8$ 的 FFT 蝶式计算图，行从 0 到 $n-1$ 进行编号，列从 0 到

* 本文研究得到高校博士学科点专项基金资助. 作者陈国良，1938 年生，教授，博士导师，主要研究领域为并行分布式算法，智能计算. 李晓峰，1971 年生，博士生，主要研究领域为并行处理. 黄伟民，1971 年生，硕士，主要研究领域为智能机器人及并行处理.

本文通讯联系人：陈国良，合肥 230027，中国科技大学计算机系

本文 1995－07－07 收到修改稿

$\log n$(本文对数以 2 为底)进行编号. 一个第 r 行,第 i 列的点将分别连到第 r 行,第 $i+1$ 列和第 r' 行,第 $i+1$ 列的点上,其中 $r'=r+n/2^{i+1}(MOD\ n/2^i)$. 在串行机上一个 n 点的 FFT 算法的时间复杂度为 $O(n\log n)$.

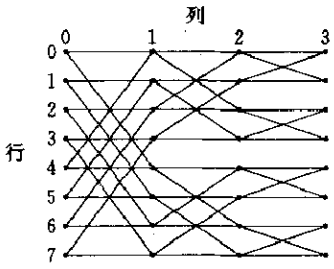

图1 $n=8$的FFT蝶式计算图

PRAM 是一种共享存储的同步计算模型,各处理器均可在单位时间内进行读、写共享内存,而它们之间的同步是在一个全局时钟下以锁步(Lockstep)方式实现的,因而在进行算法的设计和分析时同步是隐含的,无需在算法中明确设置同步点. 在 PRAM 中进行 FFT 计算时,假定 n 个点均匀地分配给 p 个处理器($n\geqslant p$),这些处理器将并行地完成各自的局部计算;可任意读、写共享存储单元(当然要遵守不同的并发读写限制,如 EREW,CREW 和 CRCW);然后同步地开始下一个局部计算. 所以在此情况下,FFT 算法的设计和分析比较简单. 然而,PRAM 模型对异步操作的 MIMD 机器显然是不适合的;同时,假定访问内存时间为单位时间也是不合适的,因为考虑到存储带宽和存储管理,此时间远比局部运算时间长. 于是异步的 PRAM 模型,即 APRAM 就被提出来了.[2]

2 APRAM 模型上的 FFT 算法

APRAM 对 PRAM 的一个最关键的改进是各处理器间按异步方式进行操作,它们可按照自己的进度进行处理,只是在一些路障(Barrier)点才相互同步等待. 这些同步点一般设置在不同的处理器需访问同一存储单元的时刻(这样就可以保证其后的操作可正确完成). 按此,在 APRAM 中,计算被分成了一些由路障同步分开的全局相(Phase). 在每相内,每个处理器均异步地运行其局部程序,局部程序中的最后一条指令是路障同步指令;各处理器均可异步地读、写全局存储器,但在同一相内不允许 2 个处理器访问同一存储单元. 因为在 APRAM 中,算法的总时间是相数乘上相内所花费的时间,所以算法设计时应尽量减少相数,而且在相内应设法将局部计算时间与路障同步时间相匹配,这也就是算法设计时所追求的目标. 假定 B 为路障同步所需的时间,则在相内应设法安排计算时间与 B 大致相当,这样才能充分利用处理器的计算能力.

根据蝶式计算的特点(参见图 2)可知,第 i 列中的节点 v 的值可以从第 $(i-k)$ 列中的 2^k 个节点计算出来($i>k, i\in(0,\log n)$). 这样我们不妨将第 i 列中的节点 v 视为二叉树的根(圆形节点),第 $(i-k)$ 列中的 2^k 个节点视为二叉树的叶子(方形节点);相似地,如果我们取 $2^k=B$,则对于第 i 列的节点,处理器可从第 $(i-\log B)$ 列中的 B 个叶节点计算出来. 所以在设计 APRAM 模型上 FFT 算法时,我们先将图 2 所示的蝶式计算图分成 $\log n/\log B$ 个 *stages*(也就是一些 *phases*),每个 *stage* 内有 $\log B$ 个列,而每个 *stage* 中最后一列上的诸节点的值,可以由前一

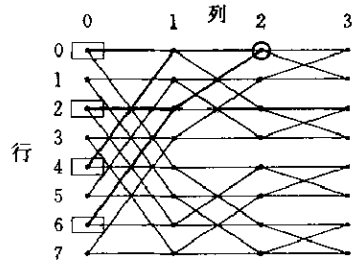

图2 APRAM模型上FFT并行算法($n=8, B=4$)

个 *stage* 中的最后一列的 B 个树叶所构成的二叉树进行计算而得到.这样,以 B 个叶节点(图 2 中的方形节点)的二叉树,其内节点(包括根)数为 $B-1$,即蝶式计算数为 $B-1$,所以当由一个处理器来完成对此二叉树的计算时,所需时间亦为 $B-1=O(B)$.以其为一个相的长度正好可满足上述所说的设计目标.按此思想所设计出的算法如下:

```
算法:APRAM 模型上的 FFT 算法
输入:n 个输入存放在全局存储器中
输出:n 点 FFT
Begin /* 每个处理器 i 均执行下列算法 */
    (1)stage←0
    (2)While stage<[logn/logB] do
        (2.1)从全局存储器中读取相应于第 i 个节点的二叉树的 B 个叶节点的值
        (2.2)用模拟二叉树的方法由 B 个叶节点计算节点 i 的值
        (2.3)将所计算的节点 i 的 B 个副本写入全局存储器
        (2.4)barrier /* 路障同步 */
        (2.5)stage←stage+1
        Endwhile
End
```

在上述算法中,为了避免在同一个相内并发访问同一单元,所以要将所计算的值的 B 个副本写入全局存储器,因为在下一个 *stage* 中 B 个处理器要读取此值.由于 While 循环中每一迭代(相应于 APRAM 的一个相)花费 $O(B)$的时间,所以在使用 n 个处理器时上述算法的总时间为 $O(B\log n/\log B)$.该算法尚可进一步改进以达到成本最优.[2]

虽然异步的 APRAM 模型对同步的 PRAM 模型有重大的改进,但 APRAM 仍是一个共享存储的模型,这与当代并行机的主流机型,即分布存储的 MIMD 巨量并行机仍不一致.为此,Valiant 又对 APRAM 模型作了进一步的改进,提出了异步分布存储的 BSP 模型.[3]

3　BSP 模型上的 FFT 算法

BSP 模型,即大同步模型,是主要面向分布存储的 MIMD 机器的,系由处理器/存储模块(简称为处理单元)、选路器和同步器组成,可用定量参数 p(处理器数),g(选路器带宽因子,即发送消息的间隔)和 L(全局同步时间间隔)描述.因为模型是分布存储的,所以专门设置了选路器,从而将计算和通信分离,以突出通信的重要性.和 APRAM 类似,在 BSP 中计算系由一系列全局同步分开的周期为 L 的超级步(Superstep)所组成.在每个超级步中,每个处理器均执行局部计算,并通过选路器接受和发送至多 h 条消息(称为 $h-relation$),然后作全局检查,以确定该超级步是否完成.在每个超级步开始,处理器仅对已经有效的局部数据施行操作,若数据未准备好,则只好推到下一个超级步的开始时再操作.在 BSP 模型中,同步周期 L(含有 L 个局部操作)至少要$\geqslant gh$,也就是说,h 至多为$[L/g]$.所以在 BSP 中设计算法时,局部计算时间应与发送/接收 h 个消息的时间大致匹配.

假定处理器数为 p,当我们在 BSP 模型上设计算法时,和 APRAM 相似,我们可以先将 n 点的蝶式计算图分成$[\log n/\log d]$($d=n/p$)个连续的 *stages*(也就是一些 *supersteps*),如图 3 所示,每个 *stage* 由 n/d 个各自独立的 d 点 FFT 蝶式图组成.这样每个 *stage* 内每个处理器有 $d\log d$ 个局部操作,即 $L=d\log d$,并且每个处理器发送和接收 d 个消息,即 $h=d$(在

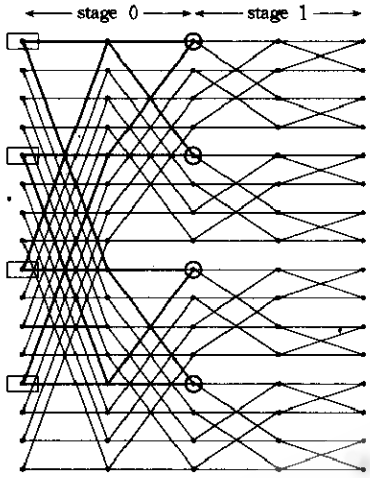

图3 BSP模型上FFT并行算法($n=16$)

执行算法时应注意,在同一个 *stage* 中,要为每条到达的消息单独分配内存区,以防不正确地更新正在计算中的数据). 所以当 $g=[L/n]=O(\log d)=O(\log(n/p))$时,算法可达到最佳,此时算法的运行时间为 $d\log d \cdot \log n/\log d = d\log n = (n/p)\log n$,而处理器数为 n/d.

由上可知,尽管 BSP 适用于分布存储从而比 APRAM 模型更加实用,但它在 2 个超级步之间仍然保留了显式同步操作. 一方面使得超级步的长度必须充分地适应任意的 h 条消息的接收和发送情况;另一方面可能有的处理器在超级步开始所发送的消息,即使网络的延迟比超级步的长度短也只好等待在下一超级步开始时才能使用,从而不能充分发挥其效率. 为了缓解此问题,Valiant 也曾提出了使用处理器子集同步的概念,即整个处理器的某些子集可不需要相互同步等待.[3]如果子集取得充分的小,那么在极端的情况下,各处理器就可采用消息传递方式完全异步地操作而不必相互等待,这实际上就演变成了下节要讨论的 Log$P$ 模型.[4]

4 Log$P$ 模型上的 FFT 算法

Log$P$ 模型是一个分布存储的异步并行计算模型,它恰巧是定量参数 L(*Latency*,消息从源到目的地的延迟),o(*Overhead*,处理器发送/接收一条消息所需的额外开销),g(*Gap*,处理器可连续发送或接收消息的最小间隔)和 P(*Processor*,处理器数)的拼写. 在 Log$P$ 模型中计算也是一系列的超级步,但与 BSP 不同:Log$P$ 中的一个超级步并非要所有的处理器都发送或接收 h 条消息;在一个超级步内消息一旦到达,处理器就可计算而不必象 BSP 那样一定要等待下一个超级步;超级步之间无显式的同步操作,每个处理器均利用点到点消息传递方式实现隐含的同步,而不必象 BSP 那样要用专门的硬件支持显式同步.

鉴于上述原因,在 Log$P$ 上进行算法设计时,要对每个处理器的局部计算步进行仔细安排,以在网络通信能力的限制范围内将通信操作和计算操作尽可能重叠起来;要妥善分配处理器,以减少对通信带宽的要求和降低远程访问的频率.

下面我们将结合 Log$P$ 模型上 FFT 算法的设计与分析来阐明上述的设计思想. 事实上,在 Log$P$ 模型上实现 FFT 算法时,首先应将 FFT 蝶式计算图中各个节点的复数计算映射到各个处理器上. 假定系统有 p 个处理器,对于 n 点($n \gg p$)FFT 而言,有下述 2 种处理器指派策略,即循环指派法和按块指派法. 在循环指派时(参照图 1 和 4),蝶式图的第 $i, i+p, i+2p, \ldots, i+n-p$ 行指派给第 i 个处理器($i=0,1,\ldots,p-1$). 此时蝶式图的前 $\log(n/p)$ 列的复数计算所需的数据都处于同一处理器中(即为局部计算),而后 $\log p$ 列中的蝶式计算所需的数据需从别的处理器中获得(即为远程计算);类似地,在按块指派时,蝶式图的前 n/p 行指派给第 1 个处理器,次 n/p 行指派给第 2 个处理器,直至指派完毕,此时前 $\log p$ 列中的计算需要远程数据,而后 $\log(n/p)$ 列中的计算是局部的. 但不管哪种指派,每个处理器都

要花费 $(n/p)\log(n/p)$ 的时间进行计算，且总有 $\log p$ 列的蝶式计算需要进行远程的读、写操作. 处理器每次发送时间为 $L+2\cdot o$，这样当发送间隔为 g 时，在 $\log p$ 列中的每个处理器总的通信时间为 $[(n/p-1)g+L+2\cdot o]\log p$，当 $g\geqslant 2\cdot o$ 时大约为 $(gn/p+L)\log p$. 所以不管上述哪种处理器指派方法，算法的总时间约为 $(n/p)\log(n/p)+(gn/p+L)\log p$，此时间显然无法使算法达到最佳. 为了在 LogP 上寻求最佳的 FFT 算法，我们可以采用处理器混合指派策略，即蝶式图的前 $\log(n/p)$ 列采用循环指派法，而后 $\log p$ 列采用按块指派法. 这样全部的计算都是局部计算，而在前 $\log(n/p)$ 列和后 $\log p$ 列之间施行一次 *remap*（参见图 4），即指派变换，实际上就是进行远程数据交换. 按此指派策略，在 LogP 模型上实现 FFT 算法可分为 3 步：①对前 $\log(n/p)$ 列，各处理器施行局部蝶式计算；②使用多对多通信方式施行各处理器之间的数据远程交换；③对剩下的 $\log p$ 列，各处理器再次施行局部蝶式计算.

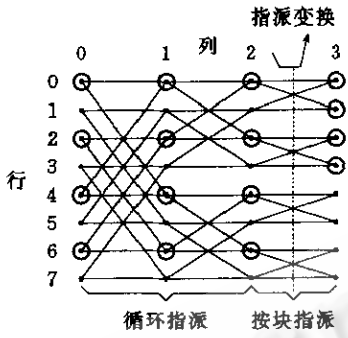

图4　处理器指派策略（$n=8$，$P=2$；带圈的节点指派给第1个处理器，不带圈的节点指派给第2个处理器）

为了尽量减少算法的总运行时间，上述处理器混合指派法是很有必要的，但这并不充分. 为此，我们还必须对各处理器中的局部计算顺序进行仔细安排，并将其与远程通信操作尽量重叠起来，这样才有可能得到较好的算法.

对于一般的并行机，总有 $L>g>o$ 的关系，所以在设计 FFT 算法时：①应使通信延迟时间 L 尽量与局部计算时间充分重叠；②为了屏蔽掉 g 的作用，应使其并入到处理器的计算时间中，这样一旦计算完成就可立即施行发送而不必再等待时间间隔 g. 为此，我们在设计 FFT 算法时，应当使每个处理器完成计算的时间相互错开，并且每个处理器一旦完成其自身的局部计算就可立即将结果发送出去，从而为其后继的计算适时准备好数据. 具体而言，在 LogP 模型上的 FFT 算法设计可分为 2 个超级步：第 1 步，循环指派处理器，各处理器进行局部蝶式计算；第 2 步，按块指派处理器，各处理器进行局部蝶式计算. 而从第 1 步到第 2 步时需要施行处理器间的数据远程通信. 为了使第 2 步中的计算能最少延迟的尽早开始，我们必须将第 1 步中的计算进行仔细安排，使得当其结束时有关节点的通信操作已基本完成，这样才能适时地为第 2 步中所需由远程通信提供的那些数据，在第 1 步中应尽可能早的计算出来，而且应考虑到尽量设法屏蔽掉时间间隔 g 的同时而尽快将其发送出去，从而不耽误第 2 步的正常计算. 参见图 5 和图 6，在此情况下，延迟时间 L 可与局部计算时间重叠，g 被隐含在节点计算时间中，而整个的开销只有参数 o，所以每个处理器的通信开销为 $(n/p-n/p^2)\cdot o$，因此算法的总时间为 $(n/p)\log(n/p)+(n/p-n/p^2)\cdot o=O((n/p)\cdot(\log n+o))$. 因为通常 $o\ll n$，所以我们在 LogP 模型上使用 p 个处理器得到了一个运行时间为 $O((n/p)\log n)$ 的近于最佳的 FFT 并行算法.

5　结　论

通过以上的分析，可对 APRAM，BSP 和 LogP 模型小结如下.

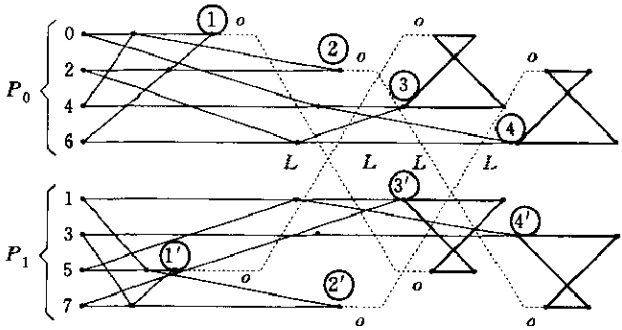

图5 Log$P$模型上FFT计算过程($n=8, p=2$)

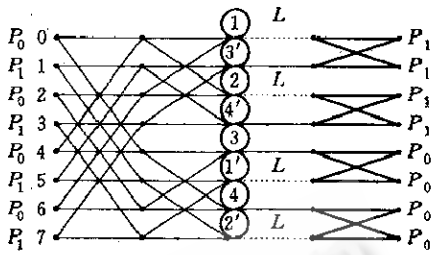

图6 重画图5为正规蝶式计算图

(1)APRAM 模型是一种共享存储的异步并行计算模型,它保留了 PRAM 可编程性的基本优点,且由于使用了显式路障同步,屏蔽了各处理器执行时间的差异,所以程序的正确性很容易保证;同时因为模型中的参数都比较容易定量计算,所以算法的分析也是不难的. APRAM 模型上算法设计的最基本出发点就是在各相内应设法使局部计算和相间的路障同步时间大致相当,以避免处理器空闲等待. APRAM 模型主要应用背景是基于共享存储的多处理器系统,但它最不实用之处是与当代主流的分布存储的并行机体系不一致,从而限制了它的发展前途.

(2)BSP 模型是一种分布存储的异步并行计算模型,基本上反映了当代并行机分布存储的主要特征,特别是选路器和处理器分离,强调了分布计算中的通信瓶颈问题;它沿袭了 APRAM 计算分相(即超级步)的朴素思想,保留了超级步之间的显式同步操作. 和 APRAM 模型类似,BSP 模型上算法设计的基本目标是力图调整超级步的长度,以能充分地适应任意的 h—relation,但这一点也正是人们最不喜欢的. 此外,在 BSP 模型的算法中所设置的全局路障同步检查点,一方面在很多并行机中可能并无相应的硬件支持;另一方面这种处理器全集同步方式可能使那些未参与消息收发操作的处理器出现空闲等待现象.

(3)Log$P$ 模型主要以近代巨量并行机为背景,吸取了 BSP 模型的很多基本思想,更充分地反映了分布存储并行机的性能瓶颈,用 L,o,g 3 个基本定量参数刻划了通信网络的基本特性,这在所提出的 3 个实用模型中算是比较多的反映了并行机的硬件特性,因而此模型也更为实际,从而倍受计算机工程技术人员的重视和欢迎. 另一方面,尽管从原理上讲,Log$P$ 模型中的超级步不必象 BSP 那样受发送或接收消息数目的限制,且超级步之间也无明显的同步操作,从而算法的设计似应更加方便和灵活,但实际上,Log$P$ 模型上算法的设计技巧性很强,不同的算法其设计技巧亦不同,至少目前尚无章可循. 因此 Log$P$ 模型最终是否能被算法界承认下来并广泛使用,还有待于大量的算法设计实例来证实,而且也尚需在使用中不断完善.

参考文献

1 陈国良. 更实际的并行计算模型. 小型微型计算机系统,1995,**16(2)**:1～9.

2 Gibbons P B. A more practical PRAM model. Proc. of ACM Symp. on Parallel Algorithms and Architectures, 1989. 158～168.

3 Valiant L G. A bridging model for parallel computation. Comm. of ACM, 1990,**33(8)**:103～111.

4 Culler D. *et al*. Log$P$: towards a realistic model of parallel computation. Proc. of 4th ACM SIGPLAN Symp. on

Principles and Practices of Parallel Programming, 1993. 1～12.

THE DESIGN AND ANALYSIS OF PARALLEL FFT ALGORITHM ON THREE PARALLEL COMPUTATIONAL MODELS

Chen Guoliang　Li Xiaofeng

(*Department of Computer Science and Technology　University of Science and Technology of China　Hefei*　230027)

Huang Weimin

(*The Hefei Institute of Intelligent Machines　The Chinese Academy Sciences　Hefei*　230031)

Abstract　This paper studies the design and analysis of parallel FFT algorithm on three parallel computational models including APRAM, BSP and Log*P* models; reveals the intrinsic properties and relationship between these computational models; comments on the utilizable and operationable properties of them.

Key words　Parallel computation models, FFT algorithm, parallel processing.