基于 FPGA 的细粒度并行 CYK 算法加速器设计与实现

夏 飞[1)] 窦 勇[1)] 宋 健[2)] 雷国庆[1)]

[1)](国防科学技术大学计算机学院 长沙 410073)

[2)](中国人民解放军 61785 部队 北京 100075)

摘 要 基于随机上下文无关文法(SCFG)理论模型进行 RNA 二级结构预测是目前采用计算方法研究 RNA 二级结构的一种重要途径.由于基于 SCFG 模型的标准结构预测算法(Coche-Younger-Kasami,CYK)巨大的时空复杂度,对 CYK 算法进行加速成为计算生物学领域一个极具挑战性的热点问题.CYK 的并行性能受限于算法多维度、非一致性的数据依赖关系和较低的计算/通信比,现有的基于通用微处理器结构的大规模并行处理方案不能获得令人满意的加速效果,并且大规模并行计算机系统硬件设备的购置、使用、日常维护的成本高昂,其适用性受到诸多限制.文中在深入分析 CYK 算法计算特征的基础上,基于 FPGA 平台提出并实现了一种细粒度的并行 CYK 算法.设计采用了对三维动态规划矩阵"按区域分割"和"逐层按列并行处理"的计算策略实现了多个处理单元间的负载均衡;采用数据预取、滑动窗口和数据传递流水线实现处理单元间的数据重用,有效解决了计算和通信间的平衡问题;设计了一种类似脉动阵列(systolic-like array)结构的主从多 PE 并行计算阵列,并在目前最大规模的 FPGA 芯片(Xilinx XC5VLX330)上成功集成了 16 个处理单元(processing elements),实验结果表明作者提出的 CYK 算法加速器结构具备良好的可扩展性.当 RNA 序列长度为 959bps,CM 模型状态数为 3145 时,与运行在 Intel 双核 E5200 2.5GHzCPU、2.0GB 主存通用计算上的 Infernal-1.0 软件相比,可获得超过 14 倍的加速效果.配置一个 FPGA 算法加速器的通用计算平台的综合处理性能与包含 20 个 Intel-Xeon CPU 的 PC 集群相当,而硬件成本仅为后者的 20%,系统功耗不到后者的 10%.

关键词 生物信息学;RNA;二级结构预测;SCFG 模型;并行 CYK 算法;FPGA;硬件加速器

中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2010.00797

Fine-Grained Parallel RNA Secondary Structure Prediction Using SCFGs on FPGA

XIA Fei[1)] DOU Yong[1)] SONG Jian[2)] LEI Guo-Qing[1)]

[1)](*Department of Computer Science, National University of Defense Technology, Changsha* 410073)

[2)](*Unit* 61785, *People's Liberation Army, Beijing* 100075)

Abstract In the field of RNA secondary structure prediction, CYK(Coche-Younger-Kasami) algorithm is one of the most popular methods using SCFG (stochastic context-free grammars) model. Accelerating SCFGs for large models and large RNA database searches becomes a challenge task in computational bioinformatics because of the $O(L^3)$ computational demands that are required. General purpose computers including parallel SMP multiprocessors or cluster systems exhibit low parallel efficiency. Furthermore, large scaled parallel computers are too expensive to be used easily for many research institutes. FPGA chips provide a new approach to accelerate CYK algorithm by exploiting fine-grained custom design. CYK algorithm shows complicated data dependence, in which the dependence distance is variable, and the dependence direction is also

收稿日期:2009-04-23;最终修改稿收到日期:2010-02-10.本课题得到国家"八六三"高技术研究发展计划项目基金(2007AA01Z106、2008AA01A201)资助.**夏 飞**,男,1980 年生,博士研究生,主要研究方向为高性能计算机体系结构和生物信息学.E-mail: xcyphoenix@nudt.edu.cn.**窦 勇**,男,1966 年生,博士,教授,博士生导师,主要研究领域为高性能计算机体系结构、高性能嵌入式微处理器、可重构计算等.**宋 健**,男,1982 年生,硕士,助理工程师,主要研究方向为信息安全与防护技术.**雷国庆**,男,1986 年生,硕士研究生,主要研究方向为高性能计算机体系结构.

across two dimensions. This paper proposes a systolic-like array structure including one master PE (Processing Element) and multiple slave PEs for fine-grained hardware implementation on FPGA. By columns and assign, tasks are partitioned to PEs for load balance. Data reuse schemes reduce the need to load energy matrices from external memory. The experimental results show a factor of more than 14 speedup over the Infernal-1.0 software for 959-residue RNA sequence and a CM model with 3145 states running on a PC platform with Intel Dual-Core 2.5GHz CPU. The computational power of the accelerator is comparable to a PC cluster consisting of 20 Intel-Xeon CPUs for RNA secondary structure prediction using SCFGs, but the hardware cost and power consumption is only about 20% and 10% that of the latter respectively.

Keywords bioinformatics; RNA; secondary structure prediction; stochastic context-free grammars model; parallel CYK algorithm; FPGA; hardware accelerator

1 引 言

随着人类基因组及多种模式生物测序项目的完成,生物信息学的研究已步入后基因组时代.序列比对和搜索已成为分子生物学领域最重要的基础性工作.目前在DNA和蛋白质序列分析领域已有许多经典的研究工具,如BLAST[1]、FASTA[2]、HMMER[3]、ClustalW[4]等.而自20世纪80年代中期具有催化性质的RNA被发现以来,RNA所起的各种重要生物化学功能如蛋白质装配、mRNA剪接和编辑、RNA与蛋白质的相互作用以及RNA干扰等引起了人们的日益关注与重视.

由于RNA的各种重要功能与其结构直接相关,并且RNA分子结构上的保守性大于序列的保守性,有时序列本身相似度很低的两个RNA分子有可能具有很相似的结构,所以仅仅采用传统的序列分析工具(如BLAST、FASTA等)无法满足对RNA结构特性研究的需求.另一方面,由于结构信息的加入大大增加了RNA序列分析的复杂性,关于RNA序列分析模型及二级结构预测方法,一直是近年来生物信息学研究的热点和难点问题.

目前,最直接的RNA结构测定方法是采用X射线衍射和核磁共振(NMR),这种方法虽然结果精确可靠但是只有在有条件的实验室环境下才能进行,并且其过程非常耗时和昂贵.因此采用计算机和数学模型预测RNA序列二级结构的方法被广泛采用,成为近年来研究的热点.目前存在三类主要的RNA二级结构预测方法:基于热力学模型的Zuker最小自由能方法[5]、基于比较序列分析模型的多序列比对方法[6]和基于随机上下文无关文法(SCFG)的结构预测方法[7].其中,SCFG理论是目前最好的RNA二级结构建模方法,在RNA二级结构预测研究领域占有重要地位,目前基于SCFG理论模型的标准比对算法为Coche-Younger-Kasami,简称CYK算法.

CYK算法用于实现单条序列与单个RNA家族的共变模型(covariance model,简称CM模型)进行比对,从而判断该RNA序列是否属于该家族并且进一步地得到该序列的二级结构.CYK算法是一种三维动态规划算法[8-10],根据矩阵填充方向不同,又可分为inside和outside两种算法,但本质上并无不同之处.CYK算法的时间复杂度为$O(KL^2+BL^3)$,空间复杂度为$O(KL^2)$,其中L是RNA序列长度,B是分支状态个数,K是CM模型中的状态数.由于巨大的时空复杂度,标准CYK算法并不适合处理较大规模的RNA序列.以LSU RNA序列为例,在单个Alpha处理器上,执行不带回溯的CYK/inside算法需要17391s[11](约4.8h),带回溯的CYK/inside算法则需要27798s(约7.72h),并且存储需求超过150GB[10],将人类全基因组中的RNA片段与目前Rfam数据库中的RNA二级结构模型进行比对需要300年[23],因此对CYK算法进行加速成为一个极具挑战性的问题.

目前人们主要采用软件并行方案基于多处理器平台如Cluster系统对CYK算法实现加速.Hill等人于1991年提出了一种按照对角线方向并行的无回溯CYK/inside算法,在7个处理器上可以获得3.5倍的加速效果[12].2005年,中国科学院Tan等人提出了对三维动态规划矩阵采用层内并行处理策

略，对矩阵进行等面积的动态任务划分，在多个处理器上并行执行. 但由于 CYK 算法中每个元素的计算量与其在矩阵中所处的位置相关，这种常规的按面积等分的任务划分策略会导致处理器负载不平衡，并且随着处理器个数的增加，计算过程中的通信开销随之增长. 算法复杂的数据依赖关系导致处理器间的数据通信粒度小，通信次数频繁，因而并行处理的效率不高. 在由 8 个节点(每个节点包括 4 个 2.4GHz AMD Opteron CPU 和 8GB 主存，一共 32 个处理器)构成的 Cluster 上执行长度为 1542 的 LSU RNA 序列的比对只能获得 19 倍的加速效果[13]，并行处理效率不到 60%. 新加坡南洋理工大学 Liu 等人提出在 PC 集群上按状态层方向流水处理的层间并行处理方案，在设计时根据元素位置与计算量之间的关系，在进行计算区域划分时采用了近似等负载的任务划分策略，即单个元素计算量较小的区域包含的列数较多，单个元素计算量较大的区域包含的列数较少，尽量让每个处理器负责计算的元素的计算量之和相等而不是元素个数相等. 研究表明，新算法的并行处理效率显著提高(在包含 20 个 Intel-Xeon 2.0 GHz CPU 的多处理器平台和包含 48 个 1.0GHz Alpha EV68 处理器的 Cluster 上执行 LSU RNA 序列的比对，可以达到 16 倍和 36 倍的加速效果[11])，但是大规模并行计算机系统硬件设备的购置、使用、日常维护的成本高昂，包含 20 个 Intel-Xeon CPU 的 PC Cluster 系统硬件成本超过 20 万元，系统功耗超过 3kW，而由 48 个 Alpha 处理器构成的 Cluster 成本超过 40 万元，系统功耗超过 6kW，只有少数机构才有实力采购使用，而非一般用户所能用得上.

近年来，FPGA (Field Programmable Gate Array)器件以其可编程特性、细粒度并行能力、丰富的计算资源、灵活的算法适应性、低硬件代价和高性能功耗比成为理想的可编程系统平台. 以目前世界上的两大 FPGA 芯片制造商的高端产品为例，Xilinx Virtex5 系列的高端产品 XC5VLX330 片内集成了 51840 个 Slice，总计 331776 个逻辑单元；288 个 BRAM，总计 10368Kb 存储容量；片内还集成了 192 个DSP 模块，可达到 105GMACS 的处理能力. Altera StratixIII 系列的高端产品 EP3SL340 集成了 135200 个自适应逻辑模块(ALM)，338000 个等价逻辑单元(LE)；1144 个 M9K 存储器模块和 48 个 M144K 存储器模块，片内存储容量超过 17208Kb；片内还集成了 576 个 18×18 乘法器模块，可以达到 300GMACS 的处理能力. 同时现有的高档 FPGA 芯片还支持 PCI-E、Ethernet、RocketIO 等多种 IO 接口，而芯片硬件成本不足 2 万元，最大功耗不足 30W. 另一方面，由于生物信息学领域经典的动态规划算法往往具备良好的位级(bit-wise)并行性，CPU 与 FPGA 相结合的生物信息学算法加速器成为了研究的热点. 目前已有许多基于 FPGA 器件对 RNA 二级结构预测方法实现硬件加速的报道，例如 2006 年 Tan 等人在 FPGA 上实现了基于最小自由能模型的细粒度并行 Zuker 算法[14]，2008 年 Arpith Jacob 实现了基于最大碱基互补配对模型的 Nussinov 算法[15,21]，但是现有的研究仅限于在 FPGA 上对二维动态规划算法的并行化，并未涉及到更高维度的 CYK 三维动态规划算法. 文献[16-17]基于 FPGA 器件实现了 CFG 模型的解析过程，但并未涉及 CYK 算法的计算过程. 最近，Moscola 等人提出了两种基于 FPGA 的细粒度 CYK 算法并行计算结构，该结构对小规模的 RNA 序列和 CM 模型可以获得大约 24 倍左右的加速比，但该结构是针对特定 CM 模型的，为特定 CM 模型定制计算逻辑将耗费大量 FPGA 逻辑资源，目前只能支持长度大约为 100bps 的 RNA 二级结构预测，这意味着 Rfam 8.0 中只有 5%的 CM 模型能够在 Moscola 的硬件平台上运行.

本文针对无回溯的 CYK/inside 算法的全部计算过程，提出了一种基于 FPGA 平台的细粒度并行算法，采用“按区域分割”和“逐层按列并行处理”的计算策略对三维动态规划矩阵的计算实现了细粒度的并行化，使得多个处理单元间的负载尽可能均衡；并在此基础上设计和实现了一种类似脉动阵列(systolic-like array)结构的主从多 PE 并行计算阵列. 阵列由一个主处理单元和多个从处理单元串联构成，只有主处理单元具备外部存储器数据载入权限. 设计通过重组单元计算顺序、CM 模型预取、数据缓存、滑动窗口和数据传递流水线等策略实现处理单元间的数据重用，从而减少对片外存储带宽的需求，实现计算和通信之间的平衡. 此外本文提出的硬件算法和电路结构与 RNA 序列和 CM 模型无关. 我们在单个 FPGA 芯片上成功集成了 16 个 PE，实验结果表明，与运行在 Intel 双核 E5200 CPU，2.0GB 主存通用计算机平台上的 RNA 结构预测软件 Infernal-1.0 相比，可获得超过 14 倍的加速效果. 配置一个 FPGA 算法加速器的计算平台的综合处理性能与包含 20 个 Xeon 2.0 GHz CPU 的 Cluster 相当，而硬件成本约为 3.5 万元，仅为后者的 20%；系统功耗不超过 200W，仅为后者的 10%.

2 CYK 算法

2.1 算法简介

CYK 算法用于实现单条 RNA 序列与 RNA 家族对应 CM(Covariance Model)模型之间的比对,以判定该序列是否属于该家族.CYK 算法考虑到了结构信息,是一个典型的三维动态规划算法.算法的输入为一条长度为 L 的 RNA 序列 $x=x_1x_2\cdots x_L$ 和一个 CM 模型.CM 模型[18]是 Eddy 和 Durbin 提出的一种进行 RNA 二级结构分析的概率模型,它利用了随机上下文无关文法(SCFG),从一组相关的 RNA 序列之间的多序列比对和一致结构中构建,以刻画 RNA 家族的结构和序列信息.CM 模型由 K 个不同的状态、状态间的连接关系以及每一个状态对应的字符生成概率(e_k)和状态转移概率(t_k)组成,而状态信息中包括当前状态的类型、编号、父/子状态的数量以及编号.CYK 算法的核心是不断地迭代计算一个截面为三角形的三维矩阵[19](如图 1 所示),迭代公式如下($1\leq i\leq j+1, 0\leq j\leq L, 1\leq k\leq K$):

$$M(i,j,k)=\begin{cases}\max\limits_{\gamma\in C(k)}[M(i,j,\gamma)+t_k(\gamma)], & S(k)\in\{D,S\}\\ e_k(x_i,x_j)+\max\limits_{\gamma\in C(k)}[M(i+1,j-1,\gamma)+t_k(\gamma)], & S(k)=MP\wedge d\geq 2\\ e_k(x_i)+\max_{\gamma\in C(k)}[M(i+1,j,\gamma)+t_k(\gamma)], & S(k)\in\{ML,IL\}\wedge d\geq 1\\ e_k(x_j)+\max\limits_{\gamma\in C(k)}[M(i,j-1,\gamma)+t_k(\gamma)], & S(k)\in\{MR,IR\}\wedge d\geq 1\\ \max\limits_{(i-1\leq mid\leq j)}[M(i,mid,k_l)+M(mid+1,j,k_r)], & S(k)=BIF\\ 0, & S(k)=E\wedge d=0\\ -\infty, & \text{其它}\end{cases}$$

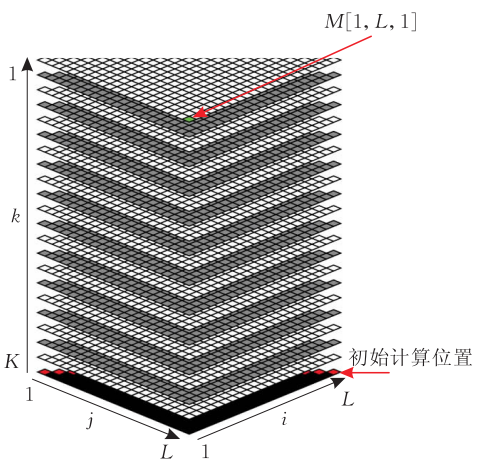


图 1 CYK/inside 算法三维动态规划矩阵的计算过程

上述公式中的 $M(i,j,k)$ 代表当前计算的元素,i,j,k 是元素在三维矩阵中的坐标.$S(k)$ 表示当前状态的类型,其中 BIF 表示分支状态,其它状态如 D,S,MP,ML,IL,MR,IR,E 统称为非分支状态.$C(k)$ 表示当前状态的子状态集合,γ 表示当前状态的某个子状态,e_k 和 t_k 分别代表当前状态下的字符生成概率和子状态转移概率,变量 $d=j-i+1$.

无回溯的 CYK/inside 算法的计算过程是从三维矩阵最下层(每一层称为一个 deck,它对应 CM 模型中的一个状态)边缘的元素开始,沿对角线按照波前(wavefront)顺序进行计算.每层元素的计算启动前首先判断 CM 模型中当前状态层的类型,根据当前状态类型选择公式中的一个分支执行.当下层 deck 所有元素计算完毕后再计算上一层的元素.计算过程由下至上逐层推进,直到 CM 模型的根节点(对应矩阵最上层的三角形)中的所有元素算完.此时元素 $M(1,L,1)$(图 3 最上层三角矩阵顶点位置的单元)的值就是最终比对分值,它代表该序列与当前 RNA 家族的相似度.

基于 FPGA 平台对 CYK 算法进行细粒度并行化同样面临一系列的挑战:(1)CYK 算法多维度、非一致性的数据依赖关系(数据相关性跨越矩阵三维空间,并且数据相关距离随元素的位置变化)导致计算过程中任务划分困难,难以实现负载平衡;(2)程序空间局部性差(访存粒度小(以 KB 为单位),每次读写操作的数据量不等且不连续),导致访存调度困难;(3)FPGA 片内存储资源不能满足矩阵 $\boldsymbol{O}(L^3)$ 的存储需求,导致片外存储器访问延时将成为提高并行处理效率的瓶颈.在 CM 模型中,三维矩阵的层数 K 大约是序列长度 L 的 3 倍,以长度为 2898bps 的 LSU RNA 序列为例,对该序列进行比对需要计算的三维矩阵的层数为 9023,每层的存储需求大约为 32MB,而目前容量最大的 FPGA 器件片内存储容量不足 2MB,因此有限的片内存储容量和相对较低的存储带宽(FPGA 工作频率不到 CPU 的 1/5)是对 CYK 算法实现硬件加速的瓶颈.

2.2 程序特征分析

通过对 CYK 算法的进一步分析,可以发现算法计算过程具备以下 4 个特征.

特征 1.　三维矩阵中每个元素的计算量与当前状态的类型和所处的位置相关.

根据 CYK 算法迭代公式，如果第 k 层为非分支层，则元素 $M(i,j,k)$ 的计算量（即加法和比较操作的执行次数）与当前状态 k 的子状态数量有关. 例如 MP 状态层元素 $M(i,j,k)$ 的计算需要执行 7 次加法和 6 次比较运算，而其它非分支状态层每个元素的计算需要执行 3～7 次加法和比较运算，具体的计算量由 CM 模型中该层对应的子状态数量确定. 而如果第 k 层为分支层，根据公式中分支层元素的数据依赖关系可知元素 $M(i,j,k)$ 的计算量 $C(i,j,k)$ 依赖于元素下标 i 和 j 的值.

$$C(i,j,k)=\begin{cases} j-i+2, & (S(k)=BIF);\\ s, s\in\{3,4,5,6,7\}, & \text{否则}\end{cases} \tag{1}$$

状态层 k 中第 j 列元素 $M(*,j,k)$ 的计算量 $C(*,j,k)$ 等于本列所有元素的计算量之和：

$$C(*,j,k)=\begin{cases}\dfrac{(j+1)*(j+2)}{2}, & (S(k)=BIF);\\ (j+1)*s,\ s\in\{3,4,5,6,7\}, & \text{否则}\end{cases} \tag{2}$$

状态层 k 中相邻两列元素的计算量之差为

$$\Delta C(j,j+1,k)=\begin{cases} j+2, & (S(k)=BIF);\\ s,\ s\in\{3,4,5,6,7\}, & \text{否则}\end{cases} \tag{3}$$

根据式(1)，可以发现非分支层元素 $M(i,j,k)$ 的计算量为常数 s，而在分支层 k 中同列元素的计算量由下往上逐渐增加，同行元素的计算量由左往右逐渐增加，其中三角矩阵右上角元素 $M(1,L,k)$ 的计算量最大，它分别依赖于其左、右子状态层的第 1 行和第 L 列，因此采用常规的等面积的任务划分策略将导致负载不平衡. 而根据式(2)和(3)，可以发现非分支状态层中相邻两列元素的计算量之差为常数 s，在分支层中相邻两列元素的计算量之差也很小，为 $O(L)$ 量级，L 为 RNA 序列的长度. 为此我们采用了按矩阵列顺序进行循环分配的任务划分策略，根据处理单元(PE)的数目将状态层矩阵划分为若干区域，每个区域包含 p 列元素（p 为 PE 的个数），在同一个区域中，每次为每个 PE 分配一列，按照列顺序依次进行计算.

特征 2.　状态层(deck)之间的数据相关距离是不确定的，它由 RNA 序列家族 CM 模型中父子节点的连接关系决定.

图 2 所示的 RNA 家族 CM 模型包含 24 个节点共 81 个状态[10]. 图中的弧线代表状态（也就是三维矩阵层与层）之间的数据依赖关系. 从图中可以看出，每一个状态（层）的计算所依赖的数据层的数量（最少为 0，最大为 6）和位置都是不确定的，数据相关距离可能为 0（即依赖于本层的数据），也可能为几十、几百乃至上千（图 2 中第 10 层的计算便依赖于第 46 层的元素）. 此外，每一个状态层的存储开销均为 $O(L^2)$，有限的 FPGA 片内存储空间无法满足 CYK 算法的存储需求. 针对该问题，我们采用了“按区域分割”和“逐层按列并行处理”的细粒度处理策略，将整个矩阵划分为多个区域，逐个区域、逐层计算，避免存储整个三角矩阵，将计算过程的存储需求由 MB 量级减少为 KB 量级，从而满足 FPGA 器件存储资源的限制.

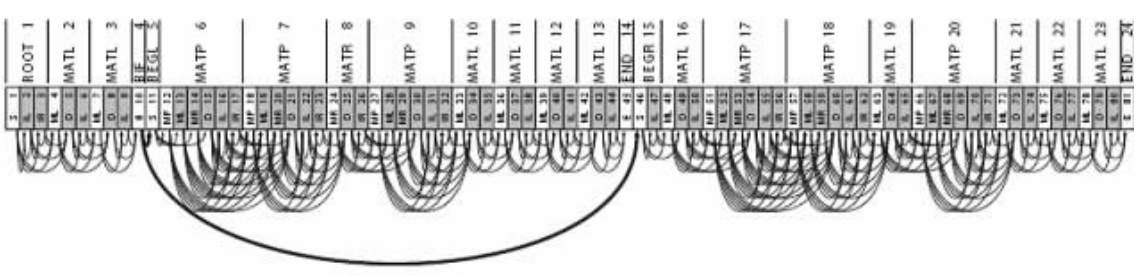


图 2　RNA 家族 CM 模型示例

特征 3.　层内(Deck)元素的数据依赖关系取决于状态类型，既可能存在于同层元素内部也可能存在于不同层的元素之间，从而导致计算过程中存在大量细粒度不连续的存储访问.

根据 CYK 算法的迭代公式，每一层元素在计算时都要根据当前状态的类型选择执行不同的分支. 对于非分支(状态)层而言，每一个数据在计算时最多依赖 6 个子状态的数据（并且这 6 个数据位于不同状态层），最少需要 1 个子状态的数据.

(1) 若当前状态 k 为 D 或 S：则元素 $M(i,j,k)$ 的计算依赖于子状态对应 Deck 中处于相同位置的元素 (i,j)；

(2)若当前状态 k 为 MP:则元素 $M(i,j,k)$的计算依赖于子状态对应 Deck 中当前位置左下方的元素$(i+1,j-1)$;

(3)若当前状态 k 为 ML 或者 IR:则元素 $M(i,j,k)$ 的计算依赖于子状态对应 Deck 中当前位置下方的元素$(i+1,j)$;

(4)若当前状态 k 为 MR 或者 IR:则元素 $M(i,j,k)$的计算依赖于子状态对应 Deck 中当前位置左侧的元素$(i,j-1)$.

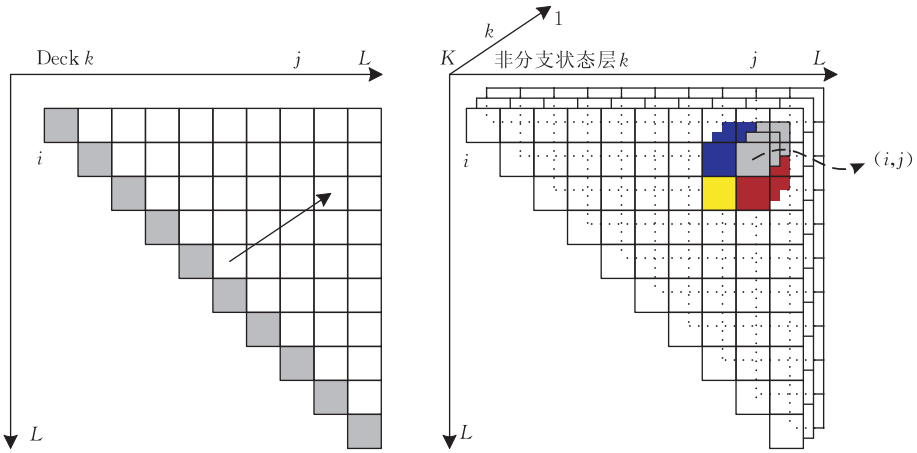


图 3 非分支状态层数据依赖关系示意图

对分支(BIF)状态层 k 而言,每个元素依赖于其左子状态层的第 i 行和右子状态层的第 j 列元素.如图 4 所示,元素 $M(i,j,k)$的值等于其左子状态同行元素 $M(i,*,k_{\text{left}})$与右子状态同列元素 $M(*,j,k_{\text{right}})$之和的最大值.根据特征 2,不管是何种类型的状态,其子状态数据层与父状态数据层的距离是不确定的,并且该数据无法存储在 FPGA 片内,所以在计算时将导致大量不连续和跳跃性的片外存储访问.

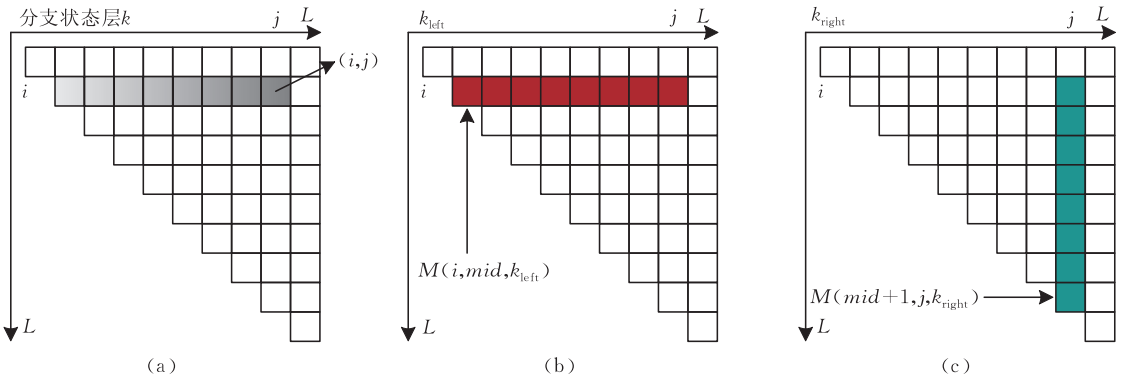


图 4 分支(状态)层数据依赖关系示意图

特征 4. 分支状态是计算的瓶颈,可以使用数据重用策略减少对访存带宽的需求.

CYK 算法时间复杂度表达式 $O(KL^2+BL^3)$中的 KL^2为非分支状态层的计算开销,BL^3为分支状态层的计算开销,两者的比例为 $KL^2/BL^3=K/BL$,一般来讲,$K\approx 3L$,并且当序列长度 L 大于 1024 时,$B\gg 3$,分支状态层的计算开销超过 90%,并且该比例随着序列长度的增加显著增加,所以分支状态层是计算的性能瓶颈.根据算法特征 3,分支状态的计算依赖于左右两个子状态上三角矩阵的行/列元素,如果按照波前计算(wavefront)策略沿对角线方向推进,则需要反复从片外载入计算所需的行/列元素.为了减少片外存储访问开销,我们采用了以下几种数据重用策略.

(1)右子状态列元素重用

从图 4 可以看到,元素 $M(i,j,k)$的计算依赖于其右子状态层第 j 列元素((c)子图用阴影填充的元素)$M(*,j,k_{\text{right}})$.对一个 PE 而言,如果采用按列顺序进行计算的策略,当计算位置从 $M(i,j,k)$移动至 $M(i-1,j,k)$时,只需要载入一个新的元素,其余

($j-i$)个列元素都可以重用，并且子状态层的计算在父状态层计算之前已经完成，如果能将本列元素缓存在片内则可避免片外存储访问.

(2)左子状态行元素重用

从图(b)可以看到，元素 $M(i,j,k)$的计算依赖于其左子状态层中的同行元素 $M(i,*,k_{\text{left}})$. 如果将 BIF 状态层第 i 行的 p 个元素(图(a)中用阴影填充的一行元素)分配到 p 个处理单元上同时执行，同时将载入的左子状态层第 i 行元素(图(b)中用阴影填充的一行元素)广播到所有的 p 个处理单元上，则可实现对载入的左子状态行元素的重用，从而减少片外存储访问开销.

(3)非分支状态元素的重用

根据图 3 所示的非分支状态元素的数据依赖关系，如果当前状态为 MP、MR 和 IR，则第 j 列元素 $M(*,j,k)$的计算依赖于子状态层中相邻的第 $j-1$ 列. 如果每个处理单元每次负责一列元素的计算，并将计算结果存储在局部存储器中，又因为子状态层的计算在父状态层计算之前已经完成，则当前状态层第 j 列元素的计算所需的数据都在 FPGA 片内(要么在本地存储器中，要么在相邻的前一个处理单元的局部存储器中)，而不需要访问片外存储器. 采用上述数据重用策略可以极大地减少计算过程中对片外子状态层数据访问的带宽需求.

3　CYK 算法加速器

3.1　系统结构

如图 5 所示，CYK 算法加速器系统平台由一台个人计算机(Host PC)和一个可重构算法加速器组成，个人计算机作为系统的主机. 系统主机负责与用户的交互，将用户输入的 RNA 序列和 CM 加载至算法加速器，并启动加速器；算法加速器完成无回溯的 CYK/inside 算法的全部计算过程，然后将比对得分报告给主机. 加速器以高性能商用 FPGA 芯片(XC5VLX330)为基础，配置两个 4GB DDRII 大容量商用存储器，通过高带宽的 PCI-E×8 数据通道与主机相连，实现通用处理器与算法加速器的协同工作. 可编程的 FPGA 芯片是算法加速器的核心，内部由 PCI-E 接口控制器、DDRII 存储控制器和 CYK 算法逻辑三部分构成.

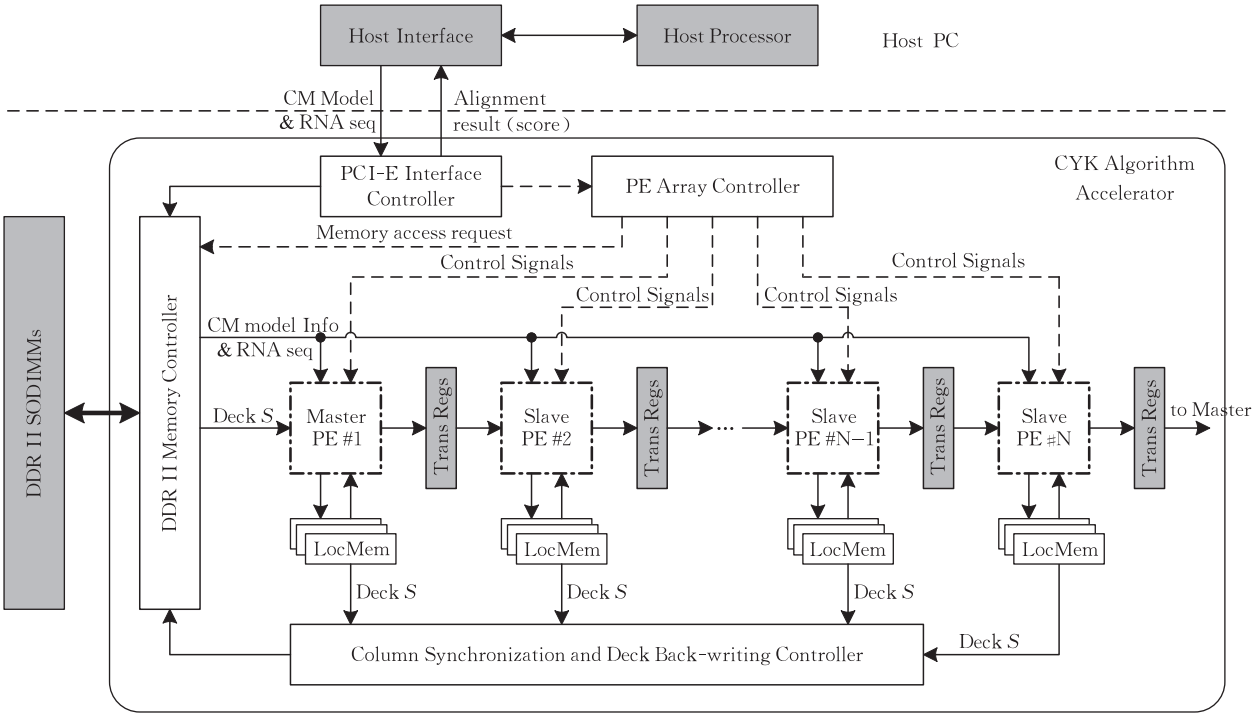


图 5　CYK 算法加速器结构

PCI-E 接口控制器基于 Xilinx IP Core 实现，负责加速器与主机间的数据通信，有效数据传输带宽可达 1GB/s. DDRII 存储控制器负责实现对片外 DDRII DRAM 的存储访问. CYK 算法逻辑包括 PE 阵列控制器、PE 阵列、列同步和写回控制器模块等部分. 其中 PE 阵列控制模块负责控制 PE 阵列的初始化、实现动态任务划分(对三维矩阵实施按列循环分配，并将其加载至 PE 阵列). PE 阵列负责三维动

态规划矩阵的并行计算，它由一系列 PE 模块串联构成，其中第一个模块是主处理单元(Master PE)，其余模块为从处理单元(Slave PE)．每一个 PE 模块都包括一个局部存储器(LocMem，采用 FPGA 片内分布式的存储资源实现，用于存储编码后的 RNA 序列)和一片计算结果缓存区，缓存区包含 16 个存储区域，采用 FPGA 片内 Block RAM 实现，最多可以缓存 16 个子状态层对应位置的一列元素．PE 间的数据传递寄存器组用于实现非分支状态的数据在 PE 阵列中的重用(根据图 3(b)所示，MP、R 状态层第 j 列元素的计算依赖于子状态层左侧 $j-1$ 列的元素)．由于每一列元素的计算量并不完全相等，列同步和写回控制器模块负责控制 PE 阵列的计算同步和将部分状态层数据写回 DRAM(如果当前状态层为左 S，需要将数据从 PE 的局部存储器(LocMem)中写回至 DRAM)．

3.2 无回溯的细粒度并行 CYK/inside 算法

标准 CYK 算法采用如图 1 所示的逐层依次计算的策略，层内采用沿对角线按照波前(wavefront)顺序进行计算．根据 CYK 算法特征 2，对当前状态层的计算而言，它所依赖的数据层的数量和位置都是不确定的，FPGA 芯片的内部存储容量不能满足 $O(L^2)$ 的存储需求；而根据 CYK 算法特征 1，采用常规的等面积任务划分策略将导致负载不平衡，所以我们采用了图 6(a)所示的采用“按区域分割”和“逐层并行处理”的计算策略．首先将整个三维矩阵沿维度 k 按列垂直切分为若干区域(section)，每一个区域仍是一个三维矩阵，它包含了每一个状态层的 n 列元素，n 为 PE 阵列的规模；然后从 $k=K$ 开始，依次计算每个区域内的所有状态层，直到 $k=1$．区域内每一个状态层内的计算采用细粒度并行的方式，每个 PE 每次负责计算当前状态层中的一列元素，当本区域当前状态层中的所有元素计算完毕后，再将下一状态层分配给 PE 阵列．如图 6(a)所示，我们以包含 4 个 PE 的处理阵列为例说明处理过程．图中的三维动态规划矩阵 $\boldsymbol{M}$ 被划分为 3 个区域

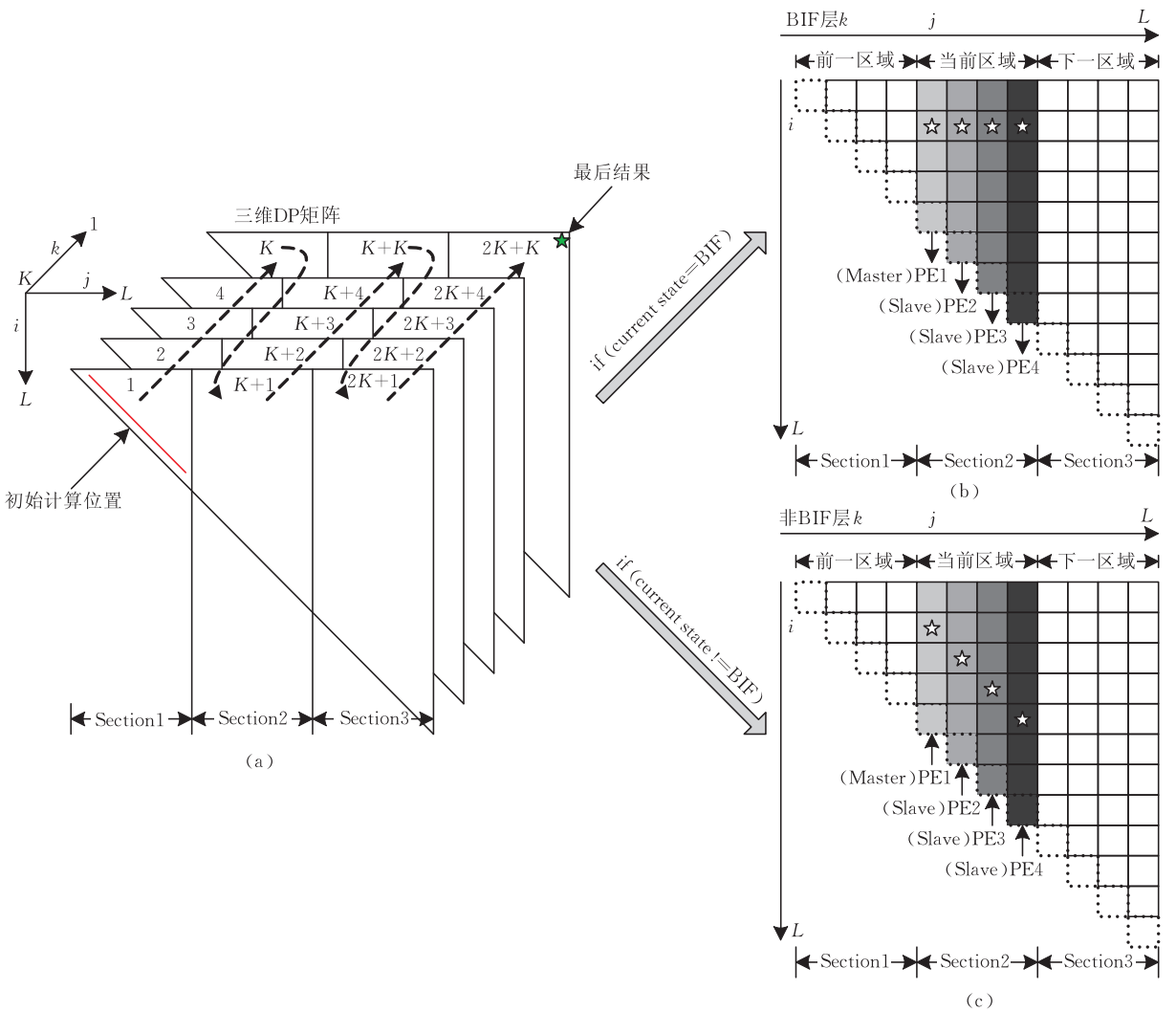


图 6 细粒度并行 CYK/inside 算法的任务划分与计算顺序

(Section1, Section2, Section3)，每个区域包含 K 层，每层包含 4 列元素. 图中标示的数字和虚线箭头代表计算顺序. 首先将 Section1 中的 $k=K$ 层加载至 PE 阵列，然后计算第 $K-1$ 层，当本区域最上层($k=1$)算完后再将 Section2 的第 K 层加载至 PE 阵列，直到最后一个区域 Section3 的最上层($k=1$)中的所有元素算完.

对每一个状态层的计算而言，首先需要根据当前状态的类型选择执行哪个不同的分支. 如图 6(b)、(c)所示，图中填充阴影的 4 列元素表示当前状态层的计算区域，它们被同时分配至 PE1～PE4 上并行计算. 图中标示五角星的单元表示 PE 阵列当前计算的元素所处的位置. 如果是非分支状态层，则每一个 PE 的计算都按照图 6(c)子图所示从当前列的底部位置开始(即三角形的斜边)，按照由下至上的顺序执行. 根据式(1)，处于同一对角线位置的元素计算量相等，所以在任意时刻，PE 阵列当前计算的元素总是处于三角矩阵的同一条对角线上. 如果是分支状态层，则每一个 PE 的计算都按照图 6(b)所示，从当前列的顶部位置开始，按照由上至下的顺序执行. 根据算法特征 3，每个 PE 负责计算区域中的一列元素，当计算位置由上至下移动时，其右子状态的列元素可以重用. 对整个 PE 阵列而言，由于都要使用左子状态的同一行元素，所以可将从 DRAM 中载入的第 i 行元素广播至所有处理单元实现行元素的重用.

图 7 采用单程序多数据模型(SPMD, Single Program Multiple Data)描述了细粒度并行 CYK/

算法 1. Fine-grained Parallel CYK/inside Algorithm without Traceback.

Input: R: an RNA sequence $R=r_1 r_2 \cdots r_L$ of length L;
　　G: a CM model of length K with states numbered in preorder traversal;
Output: $M[1,L,1]$: optimal global alignment score;
Variables: i,j,k: element location index in the three-dimensional DP-matrix;
　　$S(k)$: the type of current state k; $C(k)$: the states that k can transit to; $P(k)$: the set of parents of state k;
　　s: the number of section that the three-dimensional DP-matrix been partitioned;
　　m: current section number; p: current PE number; n: the number of PE;
　　d: the number of deck storage region in a PE local memory and $d\in\{1,2,\cdots,16\}$;
　　$M[*,j_p,k]$: the column of current state k in matrix $\boldsymbol{M}$ assigned to PE, $p\in\{1,2,\cdots,n\}$;

BEGIN: Parallel CYK/inside
　For all processing element $p\in\{1,2,\cdots,n\}$ do in parallel {
　　FOR $m=1$ up to s do {
　　FOR $k=K$ down to 1 do {
　S_1: *Get_deck_buf_id*(d); //Find a free storage region with minimal number in loc_mem;
Data input: S_2: if ($p=1$) //it's the Master PE
　S_{21}: if ($S(k)=BIF$) then {
　　Load $M[*,j_p,k_{\text{right}}]$ of the right child deck from own local on-chip memory);
　　Load $M[i,*,k_{\text{left}}]$ of the left child deck from external DRAM and *Distribute* them to all PEs};
　S_{22}: else if ($S(k)=MR$ or IR or MP) then
　　for all $\gamma\in C(k)$ do {Load $M[*,j_{p-1},\gamma]$ from external DRAM};
　S_{23}: else for all $\gamma\in C(k)$ do {Load $M[*,j_p,\gamma]$ from own local on-chip memory};
　S_3: else //it's a Slave PE
　S_{31}: if ($S(k)=BIF$) then {
　　Load $M[*,j_p,k_{\text{right}}]$ of the right child deck from own local on-chip memory);
　　Recv $M[i,*,k_{\text{left}}]$ of the left child deck from external DRAM};
　S_{32}: else if ($S(k)=MR$ or IR or MP) then {
　　for all $\gamma\in C(k)$ do {*Load* $M[*,j_{p-1},\gamma]$ from previous Processing Element $p-1$};
　S_{33}: else for all $\gamma\in C(k)$ do {*Load* $M[*,j_p,\gamma]$ from own local on-chip memory};
Calculation: S_4: if ($S(k)=BIF$) then
　　FOR $i=1$ up to $j+1$ do
　　Calculate $M(i,j_p,k)$ using the recurrence formula of Section2 and Store $M(i,j_p,k)$ in $deck\_buf[d]$;
　　else FOR $i=j+1$ down to 1 do
　　Calculate $M(i,j_p,k)$ using the recurrence formula of Section2 and Store $M(i,j_p,k)$ in $deck\_buf[d]$;
Synchronization: S_5: PE_ p waiting until the last PE_n processing finished;
Data output: S_6: if ($S(k)$ is a left substate of BIF), then send $M(*,j_p,k)$ to external DRAM;
　　else if (($p=n$) & (MR or IR or $MP\in P(k)$)), then send $M(*,j_p,k)$ to external DRAM; //for the last PE
　S_7: FOR $t=1$ up to 16 do {
　　if all the parent states of $deck\_buf[t]$ have been worked out;
　　then set $deck\_buf[t]$ free and add t into idle region set; }
Section advance:
　S_8: Distribute a new section to PE array;
　S_9: if ($k=1$) then $j_p=j_p+n$; //assign next section to PE array;
　S_{10}: if ($j_p>L$) then Stop; //current column index exceed the RNA sequence length
　};
　}; Return ($M[1,L,1]$);
}; END

图 7　无回溯的细粒度并行 CYK/inside 算法

inside 算法执行过程. 算法的输入为一条长度为 L 的 RNA 序列 $x=x_1\cdots x_L$ 和一个包含 K 个状态的 CM 模型. 算法的核心是两重 For 循环语句,循环体内部按照图 6(a)所示的任务划分策略将当前状态层中一块包含 n 列元素的区域加载至 PE 阵列进行处理. 在初始化阶段,每一个 PE 单元在本地存储器中选择一个空闲的存储区域存放将要计算出的列元素(语句 S_1),然后根据当前状态类型选择执行不同的分支. 处理过程包括数据载入(data input)、列元素计算(calculation)、列同步(synchronization)、数据输出(data output)和新区域分配(section advance)5 个步骤. 在数据载入阶段,不同类型的 PE 单元根据当前的状态采用不同的数据载入方式(S_2):只有主处理单元(Master PE,$p=1$)具备外部存储器访问权限,如果当前状态为 BIF,则 Master PE 从本地局部存储器中读取右子状态的列元素,从 DRAM 中读取左子状态的行元素并将其发送至所有 PE 单元(S_{21});如果当前状态为 MR、IR 或者 MP,则 Master PE 计算所需的数据都在片外存储器中,需要从片外存储器载入计算所需的所有子状态的第 $j-1$ 列元素(S_{22});如果当前状态为其它非分支状态,则 PE 单元计算所需的数据都在本地存储器中,只需从局部存储器中载入所需的子状态第 j 列元素即可(S_{23}). 与主处理单元的区别在于,从处理单元(Slave PE)不发出数据载入请求,它根据当前的状态类型,选择从本地局部存储器中读取数据、从前一个 PE 局部存储器中读取数据或者等待接收主处理单元读回的数据(S_3).

PE 单元的计算过程采用数据驱动的流水线模型,一旦计算所需的有效数据到达便启动该分支对应的计算流水线. 尽管所有 PE 都使用标准 CYK 算法的迭代公式,但是需要根据不同的状态类型采用不同的计算顺序:如果是非分支状态层,则按照由下至上顺序计算;如果是分支状态层,则按照由上至下的顺序执行计算(S_4). 所有 PE 的计算结果都缓存在各自的局部存储器中.

由于每一列元素的计算量不等,所以当前 PE 完成本列元素的计算后要进入列同步等待状态(PE 在同步等待的同时执行写回操作),以便保持数据依赖关系的一致性(S_5). 如果当前状态是分支状态的左子状态,则所有 PE 都需要将当前的计算结果写回片外存储器(因为分支的左子状态数据在未来很长一段时间内都不会使用). PE 阵列采用依次写回的策略,先算完先写回,尽量将写回开销隐藏在计算开销中. 如果当前状态是 MR、IR 或者 MP 状态,则只有最后一个从处理单元需要将当前的计算结果写回片外存储器(该 PE 的计算结果位于当前计算区域的最右侧,下一个区域计算时将会被用到). 除此以外,所有 PE 的计算结果都保存在局部存储器中,以便在后续的计算中使用或者通过数据传递寄存器供相邻的下一个 PE 使用(S_6). 当数据写回操作完成后,执行局部存储器数据替换操作:如果某一状态的所有父状态都已算完,则释放该状态占用的数据缓冲区(S_7).

当最后一个 PE(PE[n])计算完成并将结果写回后(说明此时本区域的计算任务已完成),PE 控制模块将本区域的下一个状态层加载至 PE 阵列(S_8),然后所有 PE 同时开始执行新区域的计算. 所有 PE 单元重复上述计算过程直到新指派的元素列号超出输入的 RNA 序列的长度(此时当前 PE 处于空闲状态,直到阵列中所有的 PE 计算完成). 当三维矩阵最后一个状态层的计算完成后,PE 阵列将最终的计算结果 $M(1,L,1)$ 写回片外存储器,计算过程终止.

3.3 PE 模块的设计

PE 模块的功能是按照 CYK/inside 递归公式实现对三维动态规划矩阵元素的计算. 主要由 PE 控制模块(PE Control Module)、分值计算模块(Computation Module)、RNA 序列存储器(Seq Mem)、CM 模型存储器(CM model info buf)和计算结果局部缓存区(Deck bufs)5 部分构成.

如图 8 所示,PE 控制模块主要包含一个控制状态机和数据 IO 通路. 控制状态机负责实现 PE 模块的初始化,从片外 DDRII 存储器中载入 RNA 序列和 CM 模型的当前状态信息;启动 PE 模块的计算,控制数据载入、分值计算、阵列同步以及数据写回和替换.

分值计算模块(图 8 中部的虚线框部分)由 END、DSPLR 和 BIF 3 个子模块构成,用于实现对应状态分支的计算. 3 个子模块的内部结构大致相同,都由一个 32 位加法器和一个 32 位比较器构成,主要区别在于子模块数据输入来源不同. END 状态不需要从外部载入数据,该状态在整个计算过程中被作为特例处理;D、S 和 L 状态的计算只需要从本 PE 的局部数据缓存区载入数据,而在 P、R 状态下,主处理单元需要从外部 DDRII 存储器载入之前的

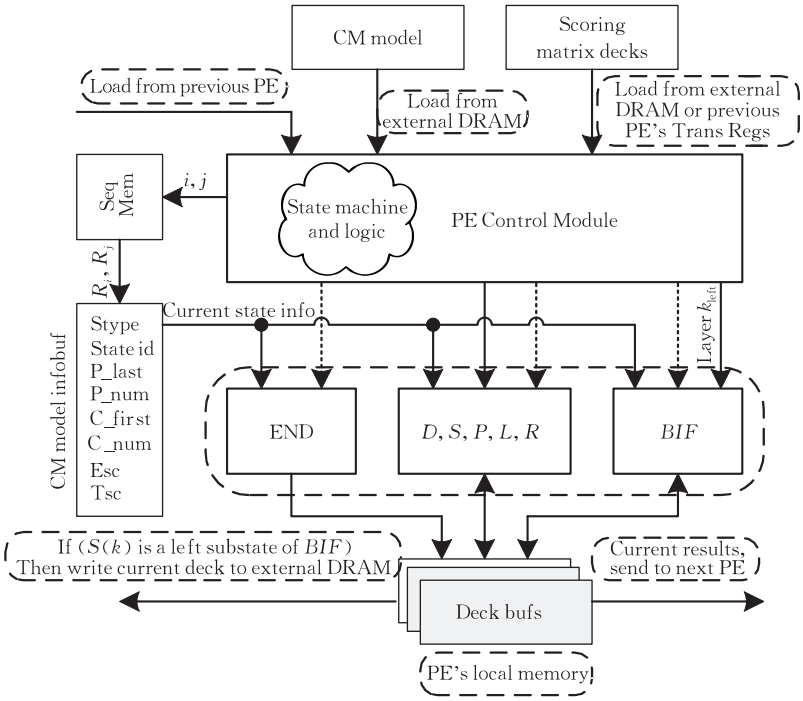


图 8　PE 模块结构

中间结果，从处理单元则从相邻的前一个 PE 载入数据；在分支状态下，BIF 计算模块需要从外部 DDRII 存储器载入左子状态层（layer k_{left}）和从自身局部缓存区中载入右子状态层（layer k_{right}）的数据. 所有子模块的计算结果都先存入局部缓存区中，当本列元素计算完成后再根据当前状态的类型判断是否需要写回 DRAM.

RNA 序列存储器（Seq Mem）用于存储当前的 RNA 序列. 系统初始化时，由 PE 控制模块将 RNA 序列从 DRAM 中取出，存入序列存储器. 相对于 FPGA 的存储容量而言，CM 模型的存储需求过大，不能将其完整地存储在片内，因此在计算过程中仅将当前状态层计算所需的信息从 DRAM 中读出缓存在 FPGA 片内. 为了实现对概率矩阵元素的并行查询，我们为每个 PE 都设置了一个 CM 模型存储器，采用 FPGA 片内分布式存储资源实现.

计算结果存储器（Deck bufs）由 16 个相对独立的存储区域（Deck buf）组成，每一个存储区可以存储三角矩阵的一列，使用 FPGA 片内存储块实现（Block RAM），用于存储本 PE 的计算结果. 局部缓存区采用双端口设计，可以直接被本 PE 和相邻的下一个 PE 访问. 假设当前 PE 的编号为 p，则 Deck bufs 中存储的是当前 section 的第 p 列元素，但这 16 列元素都位于不同的状态层. 每个存储区域都设置了两个寄存器，其中位置寄存器（deck_id）用于记录所存储的数据所处的状态层编号，父状态寄存器（p_cnt）记录还未计算出的本状态层对应的父状态数量（即本状态层元素还要用到的次数）. 当 $p\_cnt=0$ 时，则释放该存储区，并将其加入可用存储区集合. 结果缓存区的设置与 CM 模型中 BIF 状态的个数和 RNA 家族解析树结构相关，16 个缓存区已能够满足当前设计的需求.

4　实验与性能分析

4.1　并行效率分析

硬件 CYK/inside 算法加速器的执行总时间（T）等于矩阵单元的计算时间（Tc）、同步等待和写回开销（Tm）、数据载入（T_{LB}）以及列转换（section advance）开销之和，其中列转换开销只有在当前区域所有元素计算完成后才会产生，并且每次列转换只需要改变 PE 的列下标，时间开销为 $O(1)$，可以忽略，而数据载入（T_L）开销只有在分支状态下才会产生. 假设 p 是处理阵列的规模，L 是 RNA 序列的长度，B 为分支状态数量，K 为整个 CM 模型状态数量，参数 $s=[L/p]$. 由于计算过程中的并行处理方式不同，我们将分支状态和非分支状态分开考虑.

在非分支状态下，三角矩阵的计算开销（T'_{C1}）为 $T'_{C1}=(p+2p+\cdots+sp)\cdot\Delta t_1$，写回开销（$T'_{M1}$）为 $T'_{M1}=\frac{1}{2}\cdot(1+2+\cdots+s)\cdot p\cdot\Delta t_2$. 这里，$\Delta t_1=10$（为平均一个单元的计算开销），$\Delta t_2=1$（平均一个

单元的写回开销).因此,所有非分支状态层的时间开销(T_{NB})为 $T_{NB}=(T'_{C1}+T'_{M1})\cdot(K-B)$.

在分支状态下,三角矩阵的计算开销(T'_{C2})为 $T'_{C2}=\frac{1}{2}\left[\frac{s\cdot(s+1)}{2}\cdot p+\frac{s\cdot(s+1)\cdot(2s+1)}{6}\cdot p^2\right]\cdot\Delta t_3$,写回开销($T'_{M2}$)为 $T'_{M2}=\frac{L\cdot(L+1)}{2}\cdot\Delta t_4$,数据载入开销($T_{LB}$)在 PE 阵列启动当前对角线元素的计算时产生,从 master PE 发出片外存储器访问地址,到第一个元素进入 PE 阵列,每个区域需要发出$(s-1)\cdot p$次访存请求(s为当前计算的区域编号),平均每次访存需要 16 个时钟周期,所以 $T_{LB}=(p+2p+\cdots+sp)\cdot 16$.这里,$\Delta t_3=2$(执行一次加法和一次比较操作),$\Delta t_4=16$(元素按矩阵列顺序写回).因此,所有分支状态层的时间开销(T_B)为 $T_B=(T'_{C2}+T'_{M2}+T_{LB})\cdot B$.

我们可以得出总的计算开销(T_C)为

$$T_C=T'_{C1}\cdot(K-B)+T'_{C2}\cdot B \tag{4}$$

硬件 CYK/inside 算法总执行时间(包括计算和写回,按时钟周期数计算)为 T_{NB}和 T_B之和:

$$T=(T'_{C1}+T'_{M1})\cdot(K-B)+(T'_{C2}+T'_{M2}+T_{LB})\cdot B \tag{5}$$

根据式(4)和(5),CYK/inside 算法加速器的并行执行效率(E_C)为

$$E_C=\frac{T_C}{T}=\frac{1}{1+\alpha},\ \alpha=\frac{T'_{M1}\cdot(K-B)+(T'_{M2}+T_{LB})\cdot B}{T'_{C1}\cdot(K-B)+T'_{C2}\cdot B} \tag{6}$$

将 T'_{C1},T'_{C2},T'_{M1},T'_{M2}和 T_{LB}分别代入,当 $L\gg p$,$K\gg B$时,可得到 α 的近似值:

$$\alpha\approx\frac{96B\cdot p+3K}{4B\cdot L+60K} \tag{7}$$

根据式(6)和(7),以 SSU RNA 序列($L=1545$,$K=4789$,$B=30$)为例,当 $p=16$ 时,加速器的并行执行效率(E_c)接近 90%,可见细粒度并行 CYK 算法具有良好的可扩展性.

表 1 是在不同的阵列规模和输入条件下,根据式(6)计算得出的并行效率,从表中可以看出,随着 RNA 序列长度和处理器个数的增加,并行效率随之下降.我们可以从式(7)出发分析原因:

对同一条 RNA 序列而言,序列的长度 L、CM 模型状态的数量 K 以及分支状态数量 B 的值都不变,处理器个数 p 的增加将导致 α 的值增大,从而导致并行效率 E_C降低.实际上处理器个数 p 的增加导致 Master PE 的同步等待时间和数据载入开销增大,因为只有最后一个 PE 计算完成后才能给 PE 阵列分配新的计算任务;而另一方面 PE 数量增加缩短了计算时间,导致 PE 阵列计算部分占整个任务处理时间的比例下降.

表 1 CYK 算法加速器并行效率

PE 数量	RNA 序列					平均并行效率/%
	tRNA	5S rRNA	SRP RNA	RNase P	SSU rRNA	
	$B=2,L=72,K-B=228$	$B=1,L=116,K-B=356$	$B=4,L=301,K-B=923$	$B=7,L=379,K-B=1169$	$B=30,L=1545,K-B=4759$	
$p=4$	$s=18$, $E_c=91\%$	$s=29$, $E_c=94\%$	$s=76$, $E_c=93\%$	$s=95$, $E_c=93\%$	$s=387$, $E_c=95\%$	93
$p=8$	$s=9$, $E_c=87\%$	$s=15$, $E_c=92\%$	$s=38$, $E_c=91\%$	$s=48$, $E_c=90\%$	$s=194$, $E_c=93\%$	91
$p=16$	$s=5$, $E_c=83\%$	$s=8$, $E_c=90\%$	$s=19$, $E_c=88\%$	$s=24$, $E_c=86\%$	$s=97$, $E_c=89\%$	87
$p=20$	$s=4$, $E_c=81\%$	$s=6$, $E_c=89\%$	$s=16$, $E_c=86\%$	$s=19$, $E_c=83\%$	$s=78$, $E_c=87\%$	85
$p=32$	$s=3$, $E_c=78\%$	$s=4$, $E_c=85\%$	$s=10$, $E_c=82\%$	$s=12$, $E_c=78\%$	$s=49$, $E_c=82\%$	81

对相同规模的 PE 阵列和特定的 CM 模型而言,即 p、K 和 B 都不变,显然 α 的值将随着序列的长度 L 的增加而减小,所以并行效率 E_C增大.但在一般情况下,CM 模型的状态总数 K 和分支状态数量 B 都会随序列长度的增加而增大,我们将 $K\approx 3\cdot L$代入式(7)并进行变换得到

$$\alpha\approx\frac{96B\cdot\frac{p}{L}+9}{4B+180} \tag{8}$$

由于 p 不变,而 B 的值本身很小,$L\gg B$,所以 α 的值随着 L 的增大而减小,从而导致并行效率 E_C增大.实际上对不同规模的 RNA 序列而言,随着长度 L 的增加,分支状态数量也随之增加,而根据算法特征 4 的分析,分支状态层的计算开销占整个开销的 90%以上,计算开销部分增长的幅度(为$O(L^3)$量级)远大于写回和数据载入开销(为 $O(L^2)$量级),从而在 PE 阵列规模一定的情况下,对较大规模的问题能够取得更好的加速效果.

4.2 实验环境

我们在测试平台上实现了硬件 CYK 算法加速器.测试平台由一台通用计算机和一个算法加速器构成.主机配置为 Intel 双核 E5200 处理器,2.0GB 主存.算法加速器硬件主要包括 1 片 Xilinx Virtex5 系列 FPGA 芯片(XC5VLX330),两条总容量为

4GB 的 DDRII SODIMM 存储条(Kingston KVR 667D2S5/2G)，加速器通过 PCI-E×8 数据通道与主机相连. 算法加速器支持动态重构，可在 60ms 内完成不同规模的 CM 模型间的快速切换，与配置时间为秒级的常规配置方法(如 JTAG 或并行 Slect-MAP)相比，FPGA 的配置效率提高了 2～3 个数量级. CYK/inside 软件版本为 Infernal-1.0，由美国华盛顿大学医学院 Sean Eddy 实验室开发[22]，分别在 Intel E5200 双核 CPU、AMD 9650 四核以及 Intel Q9400 四核 CPU 3 种不同平台上运行.

4.3　FPGA 资源利用

我们在目前最大规模的商用 FPGA 器件 Xilinx XC5VLX330 平台上实现了硬件 CYK 算法加速器. 从表 2 的资源使用情况可以看到，在 XC5VLX330 上能够实现最大规模为 16-PE 的处理阵列，片内逻辑资源利用率为 42%，而存储资源的利用率达到了 92%，可见存储资源是系统实现的瓶颈. 从时钟频率一栏可以看到，阵列规模从 8-PE 扩大为 16-PE 时，加速器的工作频率并没有出现显著的下降，这也从实验上进一步验证了系统的可扩展性.

表 2　FPGA 资源利用情况

阵列规模	片内逻辑资源利用率	BRAM 存储利用率	频率/MHz
1-PE	16438/207360(8%)	25/288(9%)	192
8-PE	50046/207360(24%)	137/288(48%)	168
16-PE	88458/207360(42%)	265/288(92%)	164

4.4　与基于单 CPU 平台的软件 CYK 处理方案对比

(1)计算时间与加速比

我们测试了 Infernal-1.0 程序在 3 种不同的通用微处理器平台下的执行时间，并与硬件加速器进行了比较. 硬件 CYK 算法执行时间包括加速器计算时间和数据输入输出开销，FPGA 频率为 160MHz. 从表 3 可以看到，以 Intel 双核 E5200 处理器为参照，串行 Infernal 程序在 3 种计算平台上执行的性能比较接近(AMD 和 Intel 四核平台的速度大约为 E5200 CPU 的 1.1～1.3 倍)，但是 16-PE 的硬件 CYK 算法加速器性能明显超过这 3 种通用微处理器. 在算法加速器上执行 RNaseP 序列(长度为 379，模型状态数为 1176)的二级结构比对可获得接近 14 倍的加速比，而当测试序列长度为 959bps，CM 模型状态数为 3145 时，可获得 14.3 倍的加速效果.

表 3　不同平台下的软硬件算法执行时间(s)与加速比

测试序列	测试平台									
	5S rRNA $L=116$, $K=357$		SRP RNA $L=301$, $K=927$		Rnase P $L=379$, $K=1176$		Test RNA $L=959$, $K=3145$		SSU rRNA $L=1545$, $K=4789$	
	时间	加速比	时间	加速比	时间	加速比	时间	加速比	时间	加速比
Intel E5200①	0.25	1.00	5.03	1.0	8.60	1.00	279.5	1.00	798.6	1.00
AMD 9650②	0.22	1.14	4.21	1.1	7.15	1.23	227.2	1.23	614.3	1.29
Intel Q9400③	0.21	1.19	4.20	1.2	6.37	1.35	224.9	1.24	605.7	1.32
FPGA(8-PE)④	0.06	4.40	0.79	6.4	1.20	7.17	38.7	7.21	111.2	7.25
FPGA(16-PE)⑤	0.035	7.10	0.47	10.7	0.62	13.9	20.1	13.90	55.9	14.30

注：硬件环境：

① Intel Dual-Core E5200 2.50GHz CPU，2.0GB 内存；② AMD Phenom 9650 Quad CPU，2.3GHz，3.0GB 内存；③ Intel Core2 Q9400 Quad CPU，2.66GHz，3.0GB 内存；④ FPG 加速(8-PE)，160MHz，4.0GB 内存；⑤ FPGA加速器(16-PE)，160MHz，4.0GB 内存.

(2)性能功耗比

图 9 是 CYK 算法加速器与通用微处理器的性能/功耗对比. 我们同样以 Intel E5200 双核微处理器为参照，3 种通用微处理器的平均功耗约为 65W～95W，而 V5 系列中最大规模的 XC5VLX330 芯片的功耗不超过 20W，仅为 CPU 功耗的 1/3～1/5. AMD 和 Intel 四核平台的性能为 E5200 处理器的 1.3 倍，但 CPU 功耗大约是后者的 1.5 倍，所以就 RNA 二级结构预测应用而言，3 种通用微处理器的性能功耗比数据($t=1000P/W$)比较接近，而包含 16 个 PE 的 FPGA 算法加速器的性能功耗比是通用微处理器的 30 倍以上. 此外，使用电流计(型号 HIOKI3290)的测试结果表明，配置 Intel E5200 双核 CPU，2.0GB 内存的主机在运行 Infernal-1.0 程序时的平均功耗为 150W，而配置一块 FPGA 加速卡的算法加速平台功耗不超过 180W，仅增加了

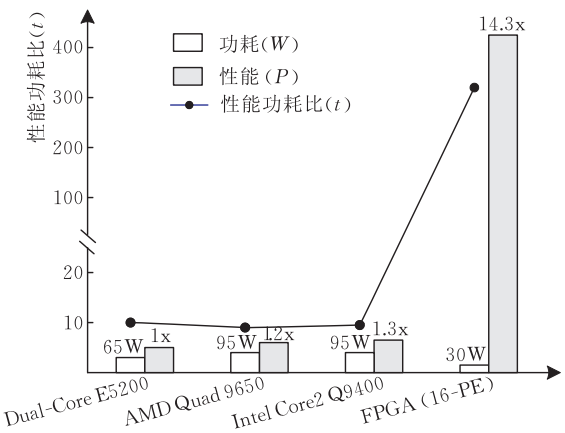


图 9　性能功耗比(FPGA vs. CPU)

20%,但获得了超过 14 倍的性能提升.

4.5 与基于多 CPU(Cluster)平台的软件并行 CYK 处理方案对比

(1)计算时间与加速比

为了与相关工作[11,13]进行比较,我们选取了两组有代表性的测试序列(RNaseP 和 SRP RNA)和对应的 CM 模型进行对比测试. Liu 和 Tan 等人分别在 Xeon Cluster[11](由 10 个节点构成,每个节点包含 2 个 Intel-Xeon 2.0GHz CPU,1GB 主存)和 AMD Cluster[13](由 8 个节点构成,每个节点包含 4 个2.4GHz AMD Opteron CPU 和 8GB 主存)平台上使用标准 C 和 MPI 函数库实现了无回溯的并行 CYK/inside 算法. 从图 10 可以看到,在这两条标准测试序列下,细粒度硬件 CYK 算法的加速效果都优于软件并行方案. 以执行 RNaseP RNA 序列的比对为例,在由 8 个 Xeon 处理器构成的 Cluster 上测得的加速比为 5.1,按照并行效率换算,在 16 个Xeon 处理器上最多可获得 7 倍的加速效果;在由 8 个 AMD 处理器构成的 Cluster 上测得的加速比为 5.9,在 16 个 AMD 处理器上可获得 10.5 倍的加速效果. 而使用包含 8-PE 的算法加速器可获得接近 7 倍的加速效果,使用包含 16-PE 的处理阵列可获得超过 14 倍的加速比. 从图中还可以看到,软件并行方案的加速性能随着处理器个数的增加而下降,而使用算法加速器可获得接近线性的加速果,显示出细粒度硬件 CYK 算法良好的可扩展性.

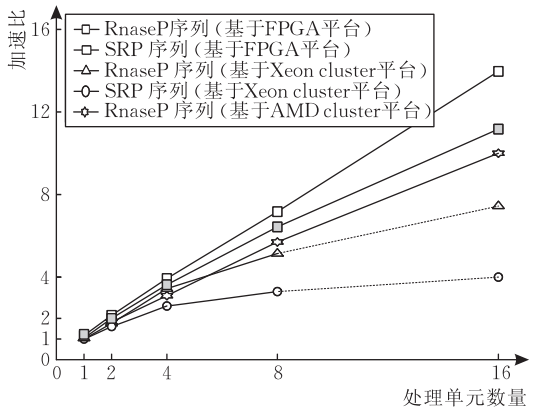


图 10 软硬件并行方案的性能对比

(2)性能价格比与性能功耗比

图 11 是 FPGA 算法加速器与基于多 CPU 的 Cluster 平台的性能、价格和功耗对比. 为了便于比较,所有指标都以包含 16 个处理器的 Xeon Cluster 为参照. 从系统造价上看,3 种 Cluster 系统的造价大约在 10～16 万元之间,而配置一个 FPGA(XC5VLX330)算法加速器的通用计算平台的价格大约为 3.5 万元,仅为 Cluster 系统的 20～30%.

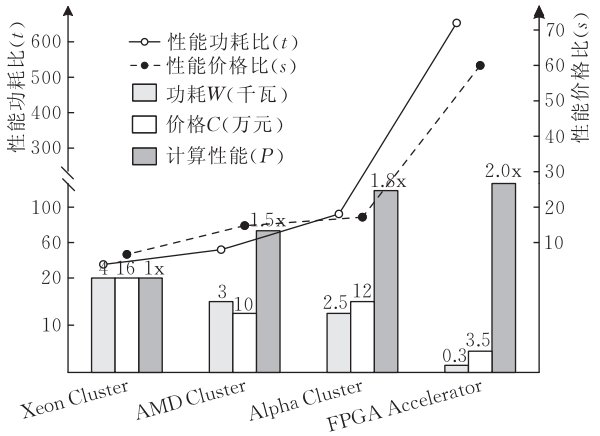


图 11 性能价格比与性能功耗比(FPGA 加速器 vs. Cluster)

从系统功耗上看,目前单个微处理器的平均功耗为 65W～ 95W[20],以平均功耗 80W 计算,由 16 个 CPU 构成的 Intel-Xeon、AMD 和 Alpha Cluster 平台的功耗大概在 2kW～3kW 之间. 而使用电流计的测试结果表明,包含 16 个 PE 的 XC5VLX330 芯片功耗不超过 20W,算法加速器的平均功耗小于 30W,整个加速器系统平台(包括主机和加速器)的总功耗不超过 200W,仅为 Cluster 系统功耗的 10% 左右.

从对 CYK 算法的整体加速效果来看,包含一个 FPGA 算法加速器的计算平台的性能与包含 20 个 Intel-Xeon CPU 的 PC 集群相当,而从性能价格比(s=100P/C)和性能功耗比(t=100P/W)参数来看,基于 FPGA 平台的细粒度并行 CYK 算法加速器方案明显优于传统的基于通用微处理器的 PC Cluster 解决方案.

5 结 论

基于随机上下文无关文法(SCFG)模型的 CYK 算法是一类重要的 RNA 二级结构预测方法,现有的基于大规模并行处理平台的 CYK 算法加速方案受限于计算通信比,随着问题规模的增长并行处理效率明显下降,此外大规模并行计算机系统硬件设备的购置、使用、日常维护的成本高昂,其适用性受到诸多限制. 本文在深入分析 CYK 算法计算特征的基础上,基于 FPGA 平台提出并实现了一种细粒度的并行 CYK 算法. 设计采用了对三维动态规划

矩阵按状态层方向进行区域分割、区域内逐层计算、状态层内按矩阵列顺序并行处理的任务划分策略，实现了处理单元间的负载平衡；采用数据预取、滑动窗口和数据传递流水线实现处理单元间的数据重用，采用数据驱动的计算流水线隐藏片外数据载入开销，达到每个时钟周期流出一个结果的效果，有效解决了计算和通信间的平衡问题；设计了一种类似脉动阵列(systolic-like array)结构的主从多 PE 并行计算阵列，并在单片 FPGA(XC5VLX330)上成功集成了 16 个 PE，实验结果表明我们提出的 CYK 算法加速器结构具备良好的可扩展性. 当 RNA 序列长度为 959bps，CM 模型状态数为 3145 时，与运行在 Intel 双核 E5200 通用微处理器上的 Infernal-1.0 软件相比，可获得超过 14 倍的加速效果. 就对 CYK 算法的加速效果而言，配置一个 FPGA 算法加速器的通用计算平台的处理性能与包含 20 个 Intel-Xeon CPU 的 PC 集群相当，而硬件成本仅为后者的 20%，系统功耗不到后者的 10%.

参　考　文　献

[1] Altschul S F, Madden T L, Schaffer A A et al. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. Nucleic Acids Research, 1997, 25(17): 3389-3402

[2] Pearson W R, Lipman D J. Improved tools for biological sequence comparison//Proceedings of the National Academy of Sciences, USA, 1988, 85: 2444-2448

[3] Database Searching Tool: HMMER[Online 2009]. Available from: http://hmmer.janelia.org/

[4] Thompson J D, Higgins D G, Gibson T J. CLUSTALW: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties, and weight matrix choice. Nucleic Acids Research, 1994, 22(22): 4673-4680

[5] Zuker M, Stiegler P. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Research, 1981, 9(1): 133-148

[6] Gutell R R, Power A, Hertz G Z et al. Identifying constraints on the higher-order structure of RNA: Continued development and application of comparative sequence analysis methods. Nucleic Acids Research, 1992, 20(21): 5785-5795

[7] Durbin R, Eddy S R, Krogh A et al. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge UK: Cambridge University Press, 1998

[8] Rivas E, Eddy S. A dynamic programming algorithm for RNA structure prediction including pseudoknots. Journal of Molecular Biology, 1999, 285(5): 2053-2068

[9] Eddy S R. What is dynamic programming? Journal of Nature Biotechnology, 2004, 22(7): 909-910

[10] Eddy S R. A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure. BMC Bioinformatics, 2002, 3: 18

[11] Tong Liu, Bertil Schmidt. Parallel RNA secondary structure prediction using stochastic context-free grammars. Concurrency and Computation: Practice and Experience, 2005, 17(14): 1669-1685

[12] Hill J C, Wayne A. A CYK approach to parsing in parallel: A case study. ACM SIGCSE Bulletin, 1991, 23(1): 240-245

[13] Tan G, Feng S, Sun N. Exploiting parallelization for RNA secondary structure prediction in cluster//Proceedings of the 5th International Conference on Computational Science. Atlanta, 2005: 979-982

[14] Tan G, Xu L, Feng S et al. An experimental study of optimizing bioinformatics applications//Proceedings of the IEEE International Parallel and Distributed Processing Symposium. Rhodes Island, Greece, 2006: 25-29

[15] Jacob A, Buhler J et al. Accelerating Nussinov RNA secondary structure prediction with systolic arrays on FPGAs//Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors. Leuven, Belgium, 2008: 191-196

[16] Ciressan C, Sanchez E, Rajman M et al. An FPGA-based coprocessor for the parsing of context-free grammars//Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines. Napa, CA, USA, 2000: 236-245

[17] Ciressan C, Sanchez E, Rajman M et al. An FPGA-based syntactic parser for real-life almost unrestricted context-free grammars//Proceedings of the IEEE International Conference on Field Programmable Logic and Application. Lausanne, Switzerland, 2001: 590-594

[18] Eddy S R, Durbin R. RNA sequence analysis using covariance models. Nucleic Acids Research, 1994, 22(11): 2079-2088

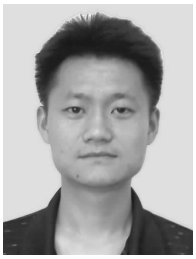
[19] Lari K, Young S J. Applications of stochastic context-free grammars using the inside-outside algorithm. Computer Speech and Languages, 1991, 5(3): 237-257

[20] CPU Power Dissipation Statistic[Online 2009]. Available from: http://en.wikipedia.org/wiki/List_of_CPU_power_dissipation

[21] Nussinov R, Pieczenik G, Griggs J R et al. Algorithms for loop matchings. SIAM Journal on Applied Mathematics, 1978, 35(1): 68-82

[22] Nawrocki E P, Kolbe D L, Eddy S R. Infernal-1.0: Inference of RNA alignments. Bioinformatics, 2009, in Press. Available from: http://infernal.janelia.org/

[23] Nawrochi E P, Eddy S R. Query-dependent banding (QDB) for faster RNA similarity searches. PLoS Computational Biology, 2007, 3(3): 0540-0554



XIA Fei, born in 1980, Ph. D. candidate. His research interests include high performance computer architecture and bioinformatics.

DOU Yong, born in 1966, professor, Ph. D. supervisor. His research interests include high performance computer architecture, high performance embedded microprocessor and reconfigurable computing.

SONG Jian, born in 1982, M. S., assistant engineer. His research interests include information security and protection.

LEI Guo-Qing, born in 1986, M. S. candidate. His research interests include high performance computer architecture and bioinformatics.

Background

RNA is an important molecule that performs a wide range of functions in biological systems. Currently, the experimental methods of determining the folded structure of an RNA molecule are time consuming and very expensive. CYK (Coche-Younger-Kasami) algorithm is one of the most popular methods using SCFG (stochastic context-free grammars) model for RNA sequence analysis. Unfortunately, the $O(L^3)$ computing requirements greatly limits the usefulness of the CYK algorithm for a sequence length of L with the explosion in gene database. General purpose computers including parallel SMP multiprocessors or cluster systems exhibit low parallel efficiency of no more than 50% resulting from complicated data dependency and tight synchronization. Recently, the use of FPGA coprocessors has become a promising approach for accelerating bioinformatics applications. Using a combination of FPGAs and general-purpose CPUs to accelerate bioinformatics application attracts much more attention. Current implementation of accelerating CYK algorithm using SCFG model can only calculate input RNA sequences with about 100 bases and the CM models with hundreds of states. This paper presents a fine-grained hardware implementation for accelerating RNA secondary structure prediction using SCFG model on FPGA. The experimental evaluation demonstrates that the performance of our accelerator is scalable with multiple PEs and that the FPGA accelerator outperforms general-purpose computers with a speedup of more than 14x on 16 PEs, however the power consumption is only about 10% of the latter. So, we believe the application-specific fine-grained scheme implemented in the accelerator provides significant advantage over the general-purpose schemes. This work is partially sponsored by the National High Technology Research and Development Program (2007AA01Z106 and 2008AA01A201).