ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.12, December 2010, pp.3094−3105
doi: 10.3724/SP.J.1001.2010.03645

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

蛋白质序列比对算法在众核结构上的并行优化*

叶笑春[1,2+], 林 伟[1,2], 范东睿[1], 张 浩[1]

[1](中国科学院 计算技术研究所 计算机体系结构重点实验室,北京 100190)

[2](中国科学院 研究生院,北京 100049)

Efficient Parallelization and Optimization of Protein Sequence Comparison Algorithm on Many-Core Architecture

YE Xiao-Chun[1,2+], LIN Wei[1,2], FAN Dong-Rui[1], ZHANG Hao[1]

[1](Key Laboratory of Computer System and Architecture, Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: yexiaochun@ict.ac.cn

Ye XC, Lin W, Fan DR, Zhang H. Efficient parallelization and optimization of protein sequence comparison algorithm on many-core architecture. *Journal of Software*, 2010,21(12):3094−3105. http://www.jos.org.cn/1000-9825/3645.htm

Abstract: In bioinformatics, a protein sequence comparison between two banks is one of most important algorithms. The sequence bank size is becoming larger and larger with the development of biotechnology, while the algorithm is also computation intensive. This leads to more and more consumption time and the single processor or multicore system, with only a few cores, are not powerful enough to reach a satisfying speed nowadays. Godson-T is a new kind of many-core architecture with lots of novel features. The parallelization of a protein sequence comparison algorithm on Godson-T is implemented. At the same time, the algorithm structure and architecture features of Godson-T are combined, and some optimization in three aspects are made: synchronization overhead, memory access contention, and load balance. The result shows that a close to linear speedup is obtained, and the performance is much better than that of the workstation platform based on the AMD Opteron processor.

Key words: sequence comparison algorithm; many-core; parallelization; optimization

摘 要: 在生物信息学中,蛋白质序列比对是最为重要的算法之一,生物技术的发展使得已知的序列库变得越来越庞大,这类算法本身又具有计算密集型的特点,这导致进行序列比对所消耗的时间也越来越长,目前的单核或者数量较少的多核系统均已经难以满足对计算速度的要求.Godson-T 是一个包含诸多创新结构的众核平台,在该系统上实现了对一种蛋白质序列比对算法的并行化,并且结合蛋白质比对算法以及 Godson-T 结构的特征,针对同步开销、存储访问竞争以及负载均衡 3 个方面对算法进行了细致的优化,最终并行部分整体也获得了更优的、接近线性的加速比,并且实际性能远远优于基于 AMD Opteron 处理器的工作站平台.

* Supported by the National Natural Science Foundation of China under Grant No.60736012 (国家自然科学基金); the National Basic Research Program of China under Grant No.2005CB321600 (国家重点基础研究发展计划(973))

Received 2008-09-23; Accepted 2009-04-27

关键词: 序列比对算法;众核;并行;优化
中图法分类号: TP301 文献标识码: A

在生物信息学中,序列比对是最基本的一类应用,目标是要找出两个序列间的相似区域.研究人员经常需要将一个新发现的基因序列和已知的基因数据库进行搜索比对,以判断该基因序列的新颖性;或者需要将两类生物体进行全基因组(full genome)序列比对,以判断它们的同源性.Smith-Waterman算法[1]是最著名的序列比对算法,它是基于Needleman-Wunsch算法[2]的改进,用于进行局部序列比对,能够得到精确的序列比对结果.Gotoh[3]对Smith-Waterman算法进行了改进,将计算复杂度从$O(M^2N)$降到了$O(MN)$.这里,M和N分别为两个比对序列的长度.但是,$O(MN)$的复杂度依然太大,难以用于大规模的基因序列比对.因此,又出现了基于启发式的算法,如BLAST[4]和FAST[5].它们首先查找出两个序列中同时存在的某个特定长度的字(word)或者种子(seed),然后根据这些字或种子所在的区域再作扩展,进行相似性的比对.启发式的算法大大缩短了运行时间,但是无法获得像Smith-Waterman算法那样精确的结果,用户需要在敏感度(sensitivity)和选择性(selectivity)之间进行调节,以获得结果精度和速度的折中.

序列比对算法计算量很大,而且当前序列数据库的增长速度已经超过通用 CPU 计算能力的发展速度,因此在大规模的序列查询中,启发式算法的速度依然难以满足实际应用的需要,对其进行并行加速显得非常必要.近年出现的片上多核/众核系统具有强大的计算能力,相比机群、MPP 等并行系统有着更高的性价比,未来的应用前景十分广泛.

本文的工作就是基于国内研究机构开发的Godson-T[15,16]众核平台来并行加速序列比对算法.

本文使用的算法是一种基于启发式的类 BLASTP 的蛋白质序列比对算法,主要用于全基因组序列比对,比对的两个序列文件均可被预先载入内存,从而避免了像数据库搜索比对中的繁重 IO 操作.但是对于较长的序列文件,比对计算仍然非常耗时,迫切需要对其进行加速.本文立足于分析蛋白质序列比对算法的算法结构,将其在 Godson-T 众核平台上进行了并行化及相关的优化工作,并获得了良好的加速效果.

本文第 1 节先简要给出蛋白质序列比对算法的流程.第 2 节介绍 Godson-T 的结构特征.第 3 节详细描述算法的并行实现,并从几个方面对算法进行了细致的优化.第 4 节给出相关的实验性能数据.第 5 节介绍与本文相关的工作;最后对全文进行总结并介绍将来可开展的工作.

1 蛋白质序列比对算法

本文所使用的蛋白质序列比对算法输入为两个序列文件,输出为序列间的相似区域,下面是一个蛋白质序列的例子:

```
I980: 112  STKIMKSAIIADSATIGKNCYIGHNVVIEDDV I IGDNS I I D---------AGTFI GRGVNIGKNA  167
           :  | |   | |:|:      |||  |   : |      |:   | ::  : :  :  |          | :  |      |||  |
P190: 261  TAKIHPSALIG----VTIGPNVVVGE--ARIQRSVLLANSQVKDHAWVKSTIVGWNSRIGKWA 320
```

这里,字符串代表由 20 个不同氨基酸组成的蛋白质序列,“|”代表序列间这两个字符完全匹配,“:”代表序列间这两个字符近似地匹配,“-“表示插入了一个空位(gap).

算法基于启发式的方法进行局部比对计算,采用子集(subset seed)[13]的方式选择种子,种子的长度设定为 4.如图 1 所示,串行算法主要分为 4 步:

(1) 将待检序列(query sequence)文件和目标序列(subject sequence)文件读入内存,对其进行索引.索引的目的是找出两个序列文件中所有的种子,并将其位置信息记录在能够快速访问到的数据结构中,如 hash 表;

(2) 对于所有的 20^4个种子(构成蛋白质的常见氨基酸有 20 种,本文选取的种子长度为 4 个氨基酸),分别找出种子在两个序列文件中的所有命中区域,待检序列文件的一个命中和目标序列文件中的一个命中组成一个“字对”,然后对这一字对所在区域进行若干个字符的无空位双向延伸(ungapped extension)比对,根据给定的置换矩阵,计算出该序列片段的分值,如果分值大于给定阈值$S1$,则记录到$T1$;

(3) 对 $T1$ 中的所有记录,对其进行带空位插入的双向延伸比对(gapped extension),查找最大的匹配序列片段,如果其分值大于给定的输出阈值 $S2$,则记录到 $T2$;

(4) 对 $T2$ 中所有结果排序输出,排序的过程中将剔除所有冗余的相同结果.

```
                                                    -Stage 1
index1=index(bank1)
index2=index(bank2)
T1=∅
T2=∅
                                                    -Stage 2
for all 20^4 possible seeds
   construct neighbouring block bk1 from index1
   construct neighbouring block bk2 from index2
                                                    -Stage 3
   for each subsequence of bk1
      for each subsequence of bk2
         compute ungapped alignment
         if score>S1
            add the result to T1
                                                    -Stage 4
for each element in T1
   compute gapped alignment
   if score>S2
      add the result to T2
                                                    -Stage 5
sort T2
for each element in T2
   display the alignment
```

Fig.1 4-Stage sequential protein sequence comparison algorithm

图 1 4 级的串行蛋白质序列比对算法

表 1 列出了当使用不同大小的序列文件进行比对时,串行算法中各个阶段所耗时的百分比.测试序列随机选取于Swiss-Prot[14](release 54.6,平均序列长度为 360 个氨基酸分子)蛋白质数据库,待检文件和目标文件序列个数相同,表中序列数量为两者之和.可以看出,随着序列的增加,Stage 2 和Stage 3 所消耗的时间占据了程序执行的绝大部分时间.

Table 1 Percentage of time spent in the various stages of the sequential algorithm

表 1 串行算法各阶段所需时间的百分比

# of sequences	100	300	1 000	3 000	10 000	30 000	100 000
Stage 1	20.2%	9.4%	3.2%	1.6%	0.3%	0.2%	0.1%
Stage 2	57.0%	68.9%	73.7%	72.5%	66.3%	64.7%	64.3%
Stage 3	18.4%	20.5%	20.8%	24.0%	30.5%	32.1%	32.6%
Stage 4	4.4%	1.2%	2.3%	2.0%	2.8%	3.0%	2.9%

2 Godson-T 体系结构

Godson-T[15,16]的结构如图 2 所示,64 个顺序双发射的微核(mini-core)组织为 2 维的MESH结构;片上集成有两级cache结构,$L1$ cache为微核内私有,$L2$ 为所有核共享,16 个bank的$L2$ 按地址低位散列在芯片四周,每侧 4 个bank并共享一个内存控制器;右下方的router连接一个支持锁嵌套的硬件锁管理器(lock manager),支持lock,unlock,barrier等语义在片上的高效实现,避免了普通加锁导致的访存操作;芯片共 4 个I/O控制器同片外交互.以上各部分具体参数见表 2.

在存储模型上,Godson-T实现了基于锁的域一致性协议[16,17],使用lock/unlock对共享变量进行显式的标示.为了高效地维护cache一致性,Godson-T实现了基于写掩码的混合写回-写穿透策略[16],在保证共享变量访问正确性的同时也获得了好的性能和扩展性.

Godson-T 还拥有一套专门针对自身硬件平台开发和优化的 runtime 系统,它实现了高效的进程管理和非抢占式的进程调度策略,并且能够提供类似 Pthread 的 API 支持,从而可以非常方便地支持程序的编写和移植.

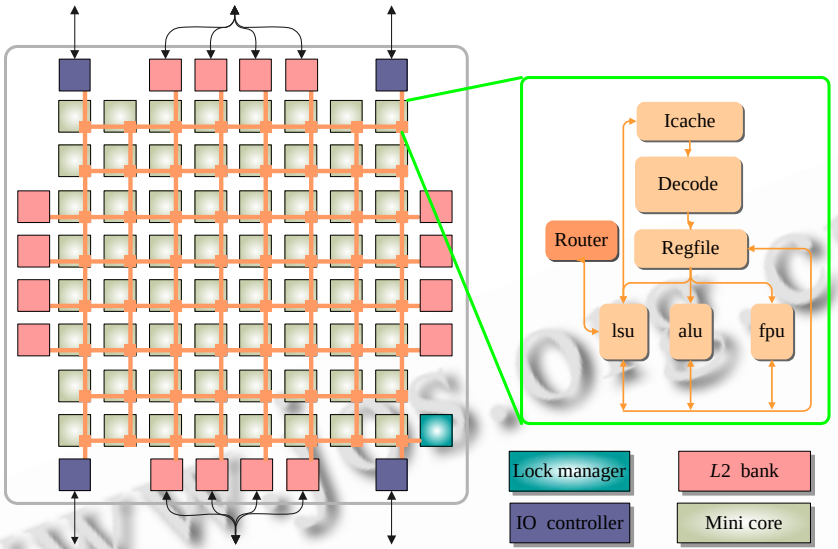

Fig.2 Godson-T architecture

图 2 Godson-T 结构示意图

Table 2 Godson-T configuration parameters

表 2 Godson-T 配置参数

Module name	Description
Mini-Core	Eight-Stage, in-order, dual-issue, mips-style mini-core; a four-stage, full-pipelined, result bypassed, floating/fixed-point unit, 192G single FLOPS with multiply-add
*L*1 dcache	Private, 32KB 4-way set associative, 32byte cacheline, 1 cycle hit latency
*L*1 icache	Private, 16KB 2-way set associative
*L*2 cache	Shared, 2MB 8-way set associative, low-order interleaved, 64byte cacheline, 12~40 cycles hit latency, 4-entry request queue in each bank, out-standing miss
Network-on-Chip	2D mesh on-chip network, with static *XY* routing strategy
Router	Two virtual channels in each direction, one entry per channel, 128-bit data, 32bit address
Memory controller	Four *L*2 banks share one memory controller, 512bit width, 1-entry request queue, 52-cycle read latency and 32-cycle write latency

3 序列比对算法的并行及其优化

本节介绍如何将上面的序列比对算法在 Godson-T 众核系统上并行化,以及相关的一些优化措施.优化的主要目标是在使用的核数不断增多的情况下,能够获得接近线性的加速比.Godson-T 的 runtime 系统实行的线程调度是非抢占式的,对于本文所讨论的算法,当创建的线程数超过物理核数时,实际性能会有所下降,故本文只讨论当线程数不超过物理核数时的加速情况.

3.1 算法并行化

表 1 的结果显示,串行算法的 Stage 2 和 Stage 3 占的时间最为明显,因此本文主要对这两步进行并行化.在串行算法中,不同种子所对应的无空位以及空位比对操作都是相互独立的,因此非常适合进行并行加速.

在Stage 1 中,为了方便对种子并行,需要快速查找到每个种子在两个序列文件中的所有命中,我们用一个hash队列和一个单向链表来记录种子的命中位置信息,如图 3 所示.单向链表项数与序列文件等长,每项对应序列文件中某个命中的起始位置,并且保存有指向下一个命中的指针,种子根据hash函数组成一个队列,队列中的

项指向种子在比对序列文件中的第1个命中位置(链表中的对应项).这样,不同的线程通过访问该数据结构就可以获得种子的字对位置信息,然后独立对其执行无空位比对、空位比对计算,如图 4 所示.为了使计算任务尽可能均匀,采用动态分配的方法将这 20^4个种子映射到不同的线程上去.

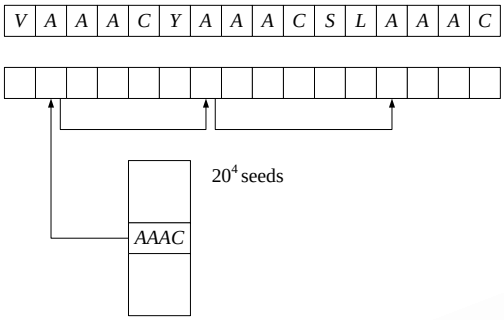

Fig.3 Structure of seed index

图 3 种子索引结构

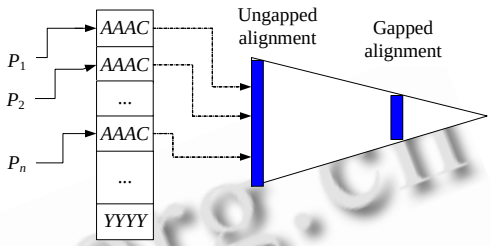

Fig.4 Parallelization of sequence comparison

图 4 序列比对的并行方式

现在我们得到并行算法 *A*(如图 5 所示),利用多线程的方式在 Godson-T 上进行加速.在算法 *A* 中,每个线程从队列中取得某个种子,然后根据其在比对序列文件中的字对位置,先进行无空位的比对计算,若超过阈值,则记录下将该成功比对的位置信息.由于不同种子之间的比对操作不需要交互,因此该信息被记录到每个线程的私有变量中.当所有种子的无空位比对都完成后,每个线程接着完成对应的空位比对计算,所有线程完成后,主线程收集各个子线程的结果,进行排序输出.

```
Main thread                                          -Stage 1
  index1=index(query sequence file)
  index2=index(subject sequence file)

  create P slave threads
  join salve threads
  merge alignment results of slave threads into T2   -Stage 4
  sort T2
  for each alignment result in T2
    display it

Slave thread[n]                                      -Stage 2
  while (seed<20^4)
    get seed s
    for each word pair of s
      compute ungapped alignment
      if (score>S1)
        add word pair to t1[n]                       -Stage 3
  for each word pair in t1[n]
    compute gapped alignment
    if (score>S2)
      add alignment result to t2[n]
```

```
Main thread                                          -Stage 1
  index1=index (query sequence file)
  index2=index (subject sequence file)

  sort all the seeds and discard the redundant one
  create P slave threads
  join salve threads
  merge alignment results of slave threads into T2   -Stage 4
  sort T2
  for each alignment result in T2
    display it

Slave thread[n]                                      -Stage 2
  while (seed<nb_seed)
    get seed s
    construct neighbouring blocks for each word pair
    for each word pair reorganized according to cache size
        compute ungapped alignment
      if (score>S1)
        add word pair to t1[n]                       -Stage 3
  for each word pair in t1[n]
    compute gapped alignment
    if (score>S2)
      add alignment result to t2[n]
```

Fig.5 Parallel version of the protein sequence comparison algorithm

图 5 蛋白质序列比对的并行算法

如图 6 所示,当线程数超过 16 时,算法 *A* 的性能略呈下降趋势,因此需要对其进行适当的优化以改善加速比.

一般说来,影响并行部分代码加速比的因素主要有 3 个:同步通信开销、存储访问竞争和负载均衡,下面分别进行讨论.

(1) 同步通信开销.并行算法 *A* 中的同步主要集中在线程从队列中获取种子后的互斥更新队列索引.随着核数的增多,锁同步所引起的竞争有加剧的趋势,见表 3 中第 2 行,数据为锁同步导致的开销占各个线程并行时间的平均百分比.

Table 3 Lock synchronization and dcache miss overhead in parallel algorithm *A* (300 vs. 300)

表 3 并行算法 *A* 的锁同步和 dcache 缺失开销(300 vs. 300)

# of threads	1	2	4	8	16	32	64
Lock overhead (%)	—	0.347	0.48	0.79	1.36	2.16	3.31
*L*1 D$ miss count	2 134 509	2 145 825	2 177 597	2 204 178	2 257 340	2 365 693	2 527 148
*L*1 D$ refill cycle	40.28	40.41	40.75	54.25	89.80	2 386.52	6 010.52

(2) 存储访问竞争.随着执行线程数的增多,假设 *L*1 cache 的缺失次数以及总的 store 操作的数量均保持不变,那么片内对 *L*2 cache 的访问密度将线性增加,在超过网络带宽上限时会导致网络拥堵,从而将对 store 操作以及 *L*1 cache 的回填都产生影响:一方面,在现有的结构下,Godson-T 的每个核内最多只能缓存 4 个 store 操作,因此在网络拥堵时,如果小核连续发出多个的 store 操作,可能无法及时送出,迫使流水线停顿;另一方面,从 *L*2 回填的时间也会明显变长,这两者都增加了并行部分的时间,从而影响程序的整体加速比.

*L*1 Icache 的缺失率很低,在 64 个线程时低于 0.001%,总的缺失次数远远低于 *L*1 Dcache 的缺失次数(不足后者的 1%).因此,*L*1 Icache 的缺失开销暂时可以忽略,而主要是放在 *L*1 Dcache 缺失以及 store 操作拥堵可能导致的开销上.

图 7 给出了因 store 操作阻塞导致的开销,最后一项为平均值.可以看到,store 导致的停顿平均占到并行执行时间(这里每个线程的并行时间是指该线程从启动到终止的时间,不包括等待其他线程结束的那部分时间.如无特殊说明,本文提到的并行执行时间均是指这个)的 23.3%,这一开销严重制约了总体的加速比.

Store 导致的网络拥堵也使得 *L*1 Dcachhe 的回填周期明显变长,表 3 的后两行给出了程序中 *L*1 Dcache 缺失的相关开销.可以看到,随着线程数的增多,cache 回填所需要的时间明显增加,到 64 个线程时,比单个线程的情况增加了 100 多倍,cache 回填的开销平均已经占到线程并行执行时间的 23.09%.

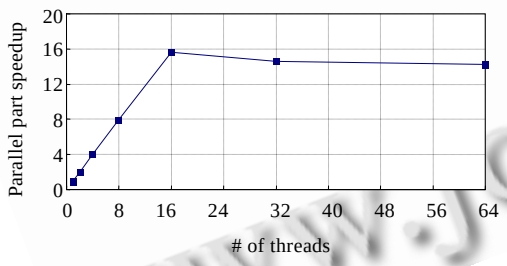

Fig.6 Parallel part speedup in algorithm *A* (300 vs. 300)

图 6 算法 *A* 的并行部分加速比(300 vs. 300)

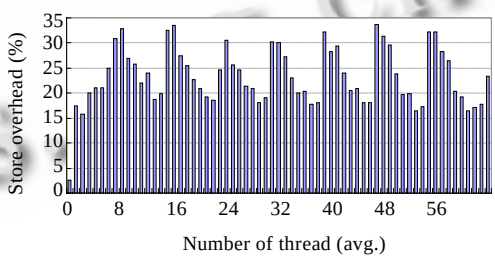

Fig.7 STORE operation overhead in algorithm *A* (300 vs. 300)

图 7 算法 *A* 中 store 操作导致的开销(300 vs. 300)

(3) 负载均衡.负载均衡与否对并行程序的加速比有着十分关键的影响,因为程序的并行执行时间总是取决于最后完成的线程.我们发现,在算法 *A* 中,大部分线程在比较接近的时间完成,但是剩余的 2~3 个线程的执行时间远远长于其他线程,这导致先完成计算任务的那些线程在退出前有大约 50%的时间都是在等待最后完成的线程.可见,负载不均衡也是制约算法 *A* 的加速比的一个重要因素.

通过对程序的分析发现,每个种子在两个序列文件中对应的字对数目是很不均匀的.少数几个种子对应的数目远远多于其他种子,字对的不均匀分布也导致了对应的非空位比对以及空位比对的不均匀分布,这些计算

任务的不均匀最终导致了不同线程之间的负载不均衡.

3.2 性能优化

本节介绍如何在同步开销、存储带宽利用以及负载均衡 3 个方面进行对应的优化工作.

在并行算法*A*中,线程获取种子时需要加锁,但是实际上,在总共的 20^4个种子中,大部分种子并不是同时在两个序列文件中均存在命中,即找不到对应的字对,这种情况称为空种子.当种子长度为 4 时,总的种子数量为 160 000(20^4).而表 4 给出了对应不同数据集的非空种子的数量,可以看到,非空种子数量均小于 5%.对于空种子,不需要进行任何的比对计算.因此,并行算法*A*中存在大量的冗余加锁操作,而且大量的空种子导致对锁的请求十分集中,很容易出现竞争.于是可以先引入一个预处理,将这些空的种子排除掉,在减小了锁的数量的同时,也降低了锁竞争的几率.

Table 4 Number of seeds with hits

表 4 非空种子数量

# of sequences	100	300	1 000	3 000	10 000	30 000	100 000
# of seeds	4 637	5 911	6 565	6 675	6 688	6 689	6 691

为了降低过多存储访问产生的竞争,首先需要尽量减少 store 的操作.对程序的剖析发现,大多数的 store 操作来自于用来执行非空位比对分值计算时的循环内的操作.代码片段如图 8(a)所示,算法需要对种子所有字对的临近区域进行比对,如果某个种子在待检序列文件中出现了 *nb_qry* 次,目标序列文件中出现了 *nb_sbj* 次,那么对应该种子总共需要进行 *nb_qry***nb_sbj* 次比对.由于比对需要针对命中区域的邻近区域作若干字符的扩展和填充处理(本文的算法是在两侧各读入 20 个字符),这部分操作有较多的访存.并且,内层循环经过 *O*(*nb_qry***nb_sbj*)次执行后,将导致实际执行指令的大量增加,更好的选择是将这部分包含大量访存的操作外移.因此,可以预先将命中的邻近区域复制到连续空间,并对到达子序列头尾等特殊情况填入替换字符,减少内层循环中的 store 数量.

```
For (i=0;i<nb_qry;i++){
    For (j=0; j<nb_sbj; j++){
        score=ComputeAlignmentScore(seq_qry,seq_sbj,i,j);
        }
}
```

(a)

```
For (k=0; k<iter; k++){
    iter_lower=k*ITER_NUM;
    iter_upper=iter_lower+ITER_NUM;
    for (i=0; i<nb_qry; i++){
        for (j=iter_lower; j<iter_upper; j++){
            score=ComputeAlignmentScore(seq_qry_c,seq_sbj_c,i,j);
            }
        }
}
```

(b)

Fig.8 Split the inner loop to improve cache performance

图 8 分割内层循环以提高 cache 性能

另外,我们还需要尽量控制总的 *L*1 dcache 缺失次数.首先,总的缺失次数少则比较不容易达到网络带宽上限,减少竞争的几率;其次,即使存在竞争,较少的缺失次数也能最低限度地降低长延时回填所带来的性能影响.前面我们将种子对应的所有命中区域的临近字符复制到连续空间,这在同时也增加了数据的局部性.并且,目标序列文件中的每个命中将被使用 *nb_qry* 次,待检序列文件中的被使用 *nb_sbj* 次,都具有良好的重用性.但是,图 8(a)的代码导致 cache 效率不高,在目前 Godson-T 中,每个节点上的 *L*1 Dcache 只有 32KB,对稍大一点的目标序

列文件,命中区域对应的字符数很容易超出 cache 容量,这会导致每个目标序列中的每个命中区域计算一次后就被换出 *L*1 cache.而在外层循环移到待检序列文件中下一个命中区域时又要重新读入,cache 中大部分数据都只使用一次.为了避免这种情况,可以对计算进行重组,将相同数据的计算任务尽量放在一起执行,以保证数据的重用性.因此,在算法中结合 Godson-T 节点上 *L*1 Dcache 的大小,对内层循环进行分割,使得连续操作的目标序列文件中的命中字符不超过 *L*1 Dcache 容量,避免反复替换.代码片段如图 8(b)所示,*ITER_NUM* 为 *L*1 Dcache 所能容纳的包含邻近区域字符的命中个数,*seq_qry_c* 和 *seq_sbj_c* 用来保存种子在待检序列文件和目标序列文件中命中区域临近字符的拷贝,而 *iter* 为每个子任务需完成的命中区域个数.假定对于每个命中,需要对比的临近字符个数为 *N*,*L*1 Dcache 的容量为 *S*,由于每个命中区域的邻近字符已经复制到连续地址空间,消除了 bank 冲突,由此得到 *ITER_NUM* 的最大值为 *S*/*N*.*ITER_NUM* 大于该值时,cache 容量不够,导致反复替换,重用率低.当 *ITER_NUM* 增大到等于 *nb_sbj* 时,将与图 8(a)中等价.*ITER_NUM* 小于 *S*/*N* 时,cache 效率也会降低,当减小为 1 时,等价于图 8(a)中内外循环交换.因此,其最优值为 *S*/*N*.实验测试得到的最优值略小于 *S*/*N*,这主要是由于在计算过程中还有一些其他数据读取访问的影响.

前面提到,线程间的负载不均衡主要是因为每个种子所对应的字对数量不均匀,我们对所有种子按照其在两个序列文件中所对应的字对数量由多到少进行排序,使得字对最多的种子可以在一开始就得到执行.而最后从队列中取下的总是字对数较少的种子,不会消耗太长的计算时间.这样可以尽可能地保证所有线程获得比较均衡的负载.这里,排序占用的是串行部分时间,但是该时间只依赖于非空种子的数量.而从表 4 中可以得知,非空种子的数量相对稳定,并不随着序列文件长度的增加而成比例增加.因此,对不同大小的比对序列文件,排序所需时间变化不大,并且远远小于 Stage 1 开销,对于大序列文件,其开销可忽略不计.

综上所述,将并行算法 *A* 作适当修改,得到了图 5 中的并行算法 *B*.

4 实验评估及结果

我们构建了一个时钟精准的、包含性能计数器的事件驱动模拟器,用来仿真 Godson-T 的实现细节.应用程序采用 gcc-3.3.3 O32-ABI 的交叉工具链编译,实验程序带-O3 选项编译,测试序列数据来自 Swiss-Prot 蛋白质数据库(release 54.6),待检序列文件和目标序列文件分别含有 300 个序列.

在排除空种子导致的冗余锁操作后,锁同步导致的开销占总的并行执行时间比例大幅下降,见表 5 第 2 行所示,它对整体加速比的影响已经很小.

Table 5 Lock synchronization and dcache miss overhead in algorithm *B* (300 vs. 300)

表 5 算法 *B* 的锁同步和 dcache 缺失开销(300 vs. 300)

# of threads	1	2	4	8	16	32	64
Lock overhead (%)	—	0.00	0.007	0.017	0.019	0.09	0.41
*L*1 D$ miss count	1 627 540	1 635 109	1 659 768	1 705 716	1 709 636	1 773 417	1 870 876
*L*1 D$ refill cycle	34.30	34.31	34.36	34.98	35.54	45.11	109.1
*L*1 D$ overhead (%)	0.610	0.611	0.622	0.650	0.661	0.863	1.98

Store 操作的数量也减少了 66%以上.图 9 给出了 store 导致小核流水线停顿的开销,最后一项是平均值.可以看到,store 导致的阻塞开销下降到平均只占并行时间的 5.52%.

表 5 中的第 3 行、第 4 行分别给出了 *L*1 Dcache 的缺失次数和平均每次回填所需的周期数.由于命中区域被拷贝到连续的地址空间,cache 缺失次数有比较明显的减少,相比表 3 中数据平均减少了 24.2%.Store 数量的大幅下降极大地缓解了片上网络压力,因此减少了 Cache 行回填所需的周期数,*L*1 Dcache 缺失导致开销所占的比例被控制在比较小的范围内.图 10 和图 11 将其与算法 *A* 得到的数据进行了对比.表 5 最后一行给出了 Cache 缺失所消耗的 cycle 数占总共并行执行时间的百分比,可以看到,在 64 个线程下,这个开销已经不到 2%.

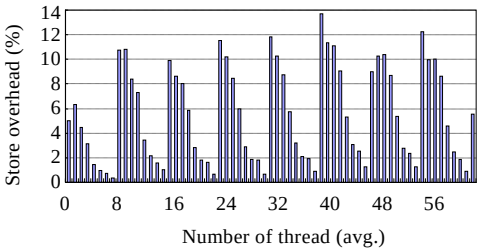

Fig.9 STORE overhead in algorithm *B* (300 vs. 300)

图 9 算法 *B* 的 store 开销(300 vs. 300)

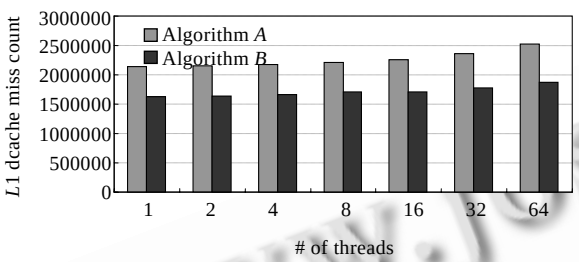

Fig.10 Comparison of *L*1 Dcache miss counts

图 10 *L*1 Dcache 缺失次数对比

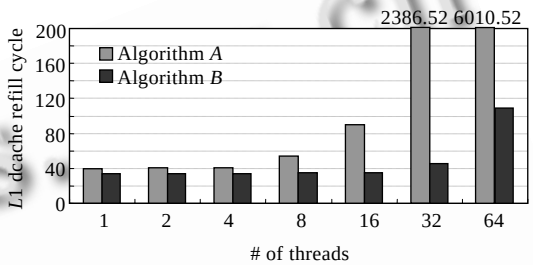

Fig.11 Comparison of average *L*1 Dcache refill cycles

图 11 *L*1 Dcache 平均回填周期数对比

下面再观察所有线程的退出时间,在 64 个线程下,平均的退出时间与最晚线程的退出时间之差只占整个并行执行时间的 2.63%;最早和最晚退出的线程时间差也仅占并行执行时间的 3.11%.因此,线程间的负载也已经变得比较均衡.

由于锁同步开销、访存操作竞争以及负载均衡问题均得到了比较好的解决,并行算法 *B* 在 Godson-T 上取得了接近线性的加速效果(如图 12 所示).

为了更直观地显示基于 Godson-T 众核平台的加速效果,我们将其与基于 AMD Opteron 265 的双 CPU 工作站的加速效果进行对比.Opteron 265 为双核芯片,运行频率 1.8GHz,片上 2M 二级 cache,工作站运行操作系统为 Red Hat Enterprise Linux Server release 5,应用程序使用 GCC 4.1.1 版,带-O3 选项进行编译.Godson-T 的频率也设定为 1.8GHz,启动 64 个线程分别在 64 个微核上执行,程序同样经过-O3 优化编译.我们对并行部分(Stage 2 和 Stage 3)的运行时间和性能进行比较(见表 6),可以看出,在相同的运行频率下,在 Godson-T 众核平台上的算法性能均可以远远优于 AMD Opteron 的 4 核(双 CPU,每个 CPU 为双核结构)工作站平台,对于不同的测试序列,分别有 4.2~4.5 倍的性能优势.这主要得益于 Godson-T 拥有众多的计算小核,并且算法经优化后具有良好的并行性和可扩展性.

最后,本文给出程序整体的加速比,包含并行部分的 Stage 2 和 Stage 3 以及串行部分的 Stage 1 和 Stage 4,结果如图 13 所示,分别为使用 600 个和 100 万个蛋白质序列比对的结果.对于小数据集,Stage 1 的影响比较明显,这限制了其整体加速比;当数据集增大时,Stage 1 可以忽略不计,但此时 Stage 4 将成为整体性能提升的瓶颈,因为 Stage 4 的拥有与 Stage 2 和 Stage 3 一样的复杂度,其所占百分比并不会随着序列文件大小的增加而降低.为了获得更好的整体加速比,Stage 4 无疑将成为下一个需要优化的目标.

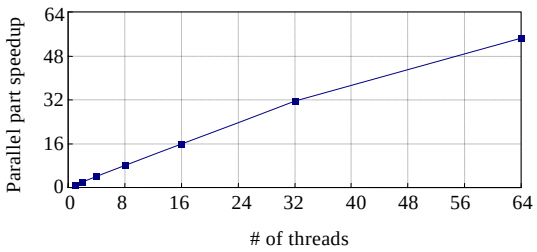

Fig.12 Speedup of the parallel part (Stage 2 and Stage 3)

图 12 并行部分加速比(Stage 2 和 Stage 3)

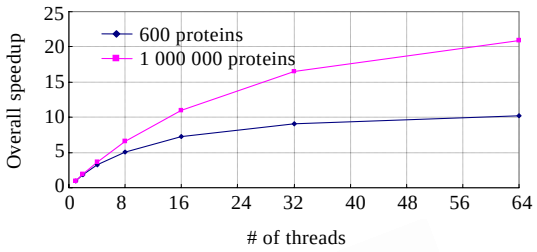

Fig.13 Overall speedup

图 13 整体加速比

Table 6 A performance comparison of Godson-T and AMD Opteron processor workstation

表 6 Godson-T 和基于 AMD Opteron 的工作站性能对比

Datasets	Cell number (G)	Godson-T (64 threads)		AMD Opteron×2	
		Time (s)	GCUPS	Time (s)	GCUPS
300 vs. 300	14.64	0.089 5	163.57	1.59	36.83
300 vs. 1000	48.09	0.290 6	165.49	4.91	39.18
1000 vs. 1000	134.66	0.777 3	173.24	13.73	39.23

(GCUPS: Giga cell update per second)

5 相关工作

由于序列比对算法计算量很大,到目前为止出现了许多对其进行加速的方法,这里既有对 Smith-Waterman 基本算法的加速,也有对 BLAST 这样的启发式算法的加速.

对Smith-Waterman算法的加速,由于受到其数据依赖关系的限制,已有的效率较高的实现大多是利用FPGA/脉动阵列等专门硬件平台进行加速,如BISP[21],SAMBA[6]以及文献[18]等.此外,也有一些基于通用平台的工作,如Rognes[22]和Farrar[23]利用Intel的SIMD指令对算法进行加速,它们实现代价比较小,但是他们均对算法本身做了一些人为的约束.而文献[19,24,25]则在多处理器系统上对Smith-Waterman算法进行了并行加速.

Blast这类启发式算法的加速实现有Mercury[7],RC_BLAST[26],RDisk[20]等专门的FPGA硬件加速系统,更多的类似实现方法可以参见文献[9]的总结.此外,还有借助现有的通用计算平台,如Intel/AMD的多核多线程方式进行加速,或者利用机群系统进行并行化,如mpiBLAST[8].而文献[27]则是利用IBM的Blue Gene这样的MPP系统对Blast进行并行加速,它们也都能获得良好的加速效果.

近年来,片上多核/众核系统开始被学术界和产业界广泛关注,相关的系统也逐渐出现,如Intel发布的万亿次级别的 80 核芯片[10]、IBM的Cell[11]以及Cyclops-64[12],中国科学院计算技术研究所正在开发的Godson-T众核芯片以及国防科学技术大学开发的MASA[28]流处理芯片等.但是我们发现,目前还极少有基于这些新的片上多核/众核系统的Blast算法的加速实现.

总体来说,在已有的诸多实现方法中,FPGA 的实现加速效果比较好,但是代价很大,并且灵活性差,难以适应算法的变化.通用 CPU 虽然价格较低,普及性高,但是加速效果比较有限.而且,通用 CPU 的发展速度已经落后于序列数据库的增长速度,依靠通用 CPU 的多核加速已经是力不从心.而商用的机群系统或者 MPP 系统则价格比较昂贵,难以大规模普及.与之相比,本文所探讨的在片上众核系统上的实现代价要小得多,并且上一节的实验结果显示,优化后的算法具有良好的扩展性,能够充分利用上众核平台上大量的计算节点,其性能也明显优于基于通用多核 CPU 的实现.因此,在这种平台上的算法并行加速是非常有吸引力的.

6 总结及未来的工作

本文介绍了一种计算密集的类 BLASTP 的蛋白质序列比对算法在 Godson-T 众核平台上的并行与优化工作.结合算法与 Godson-T 的结构特征,从 3 个方面进行了优化:(1) 通过预处理去除空种子,排除了大部分的冗余

同步操作;(2) 通过减少 store 操作的数量,减轻了片内网络压力,同时还增加了数据的局部性;并且结合 cache 大小进行计算的重组,增加了数据的重用性,减少了 cache 缺失,在很大程度上缓解了访存操作竞争带来的影响;(3) 通过排序操作对任务进行重新组织,保证了在使用较多核的时候也能获得良好的负载均衡.优化后,算法的并行部分获得了比较接近线性的加速比.这既得益于 Godson-T 高效灵活的结构实现和调度策略,也表明了算法在众核结构上具有良好的适用性和扩展性.

今后,我们还希望结合众核结构中每个小核对 SIMD 的支持来在单个核内对算法进行更细粒度的并行,比如算法中无空位插入的比对所占的计算量最大,考虑到大多数无空位比对得到的最终分值都不超过 255,只需要 8 位表示,这样 64 位的运算部件可以同时执行 8 个字对的比对计算,进一步减少了单个核的执行时间,这种粗粒度和细粒度相结合的并行方法可以获得更好的峰值速度.我们还希望将该算法在 GPU 上进行加速,特别是利用 NVIDIA 的 CUDA 编程环境进行尝试.

需要注意的是,本文所讨论的方法并不局限于针对蛋白质序列的比对,生物信息学中的其他的多种序列比对算法都可以类似地进行并行化加速.

References:

[1] Smith TF, Waterman MS. Identification of common molecular subsequences. Jounal of Molecular Biology, 1981,147(1):195−197.

[2] Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Jounal of Molecular Biology, 1970,48:443−453.

[3] Gotoh O. An improved algorithm for matching biological sequences. Jounal of Molecular Biology, 1982,162:705−708.

[4] Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. Nucleic Acids Research, 1997,25(17):3389−3402.

[5] Pearson WR, Lipman DJ. Improved tools for biological sequence comparison. Proceedings of the National Academy of Sciences, 1988,85(8):2444−2448.

[6] Guerdoux P, Lavenier D. Samba: Hardware accelerator for biological sequence comparison. Computer Application in BIOSciences, 1997,13(6):609−615.

[7] Krishnamurthy P, Buhler J, Chamberlain R, Franklin M, Gyang K, Lancaster J. Biosequence similarity search on the mercury system. In: Proc. of the 15th IEEE Int'l Conf. on Application-Specific Systems, Architecture and Processors (ASAP 2004). 2004. 365−375. http://www.computer.org/portal/web/csdl/doi/10.1109/ASAP.2004.10011

[8] Darling AE, Carey L, Feng W. The design, implementation, and evaluation of mpiBLAST. In: Proc. of the Int'l Conf. on Linux Clusters: The HPC Revolution. 2003. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.3974

[9] Lavenier D, Giraud M. Bioinformatics applications. In: Proc. of the Reconfigurable Computing. Springer-Verlag, 2005. 157−182.

[10] Vangal S, Howard J, Ruhl G, Dighe S, Wilson H, Tschanz J, Finan D, Lyer P, Singh A, Jacob T, Jain S, Venkataraman S, Hoskote Y, Borkar N. An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS. In: Proc. of the 2007 Int'l Solid-State Circuits Conf. 2007. 97−99. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?&arnumber=4242283&abstractAccess=no&userType=inst

[11] Gschwind M, Hofstee HP, Flachs B, Hopkins M, Watanabe Y, Yamazaki T. Synergistic processing in Cell's multicore architecture. IEEE Micro, 2006,26(2):10−24.

[12] Cuvillo D, Zhu W, Hu Z, Gao GR. TiNy threads: A thread virtual machine for the cyclops64 cellular architecture. In: Proc. of the 19th IEEE Int'l Parallel and Distributred Processing Symp. 2005. 265−272. http://www.computer.org/portal/web/csdl/doi/10.1109/IPDPS.2005.434

[13] Kucherov G, Noe L, Roytberg M. A unifying framework for seed sensitivity and its application to subset seeds. In: Proc. of the 5th Int'l Workshop on Algorithms in Bioinformatics. 2005. 251−263. http://www.springerlink.com/content/72384x40t0v2j254/

[14] http://www.expasy.ch/sprot/, 2008.

[15] Tan GM, Fan DR, Zhang JC, Russo A, Gao GR. Experience on optimizing irregular computation for memory hierarchy in manycore architecture. In: Proc. of the PpoPP 2008. 2008. 279−280. http://portal.acm.org/citation.cfm?id=1345255

[16] Lin W, Ye XC, Song FL, Zhang H. Using write mask to support hybrid write-back and write-through cache policy on many-core architectures. Chinese Journal of Computers, 2008,31(11):1918−1928 (in Chinese with English abstract).

[17] Iftode L, Singh J, Li K. Scope consistency: A bridge between release consistency and entry consistency. In: Proc. of the 8th Annual ACM Symp. on Parallel Algorithms and Architectures. 1996. 277−287. http://portal.acm.org/citation.cfm?id=237502.237567

[18] Wang D, Tang ZM. The implementation and analysis of Smith-Waterman algorithm on systolic array. Chinese Journal of Computers, 2004,27(1):12−20 (in Chinese with English abstract).

[19] Zhang F, Qiao XZ, Liu ZY. Parallel divide and conquer bio-sequence comparison based on Smith-Waterman Algorithm. Science in China Series E, 2004,34(2):190−199 (in Chinese).

[20] Lavenier D, Guyetant S, Derrien S, Rubini S. A reconfigurable parallel disk system for filtering genomic banks. In: Proc. of the Engineering of Reconfigurable Systems and Algorithms (ERSA 2003). 2003. 154−166. http://www.irisa.fr/symbiose/lavenier/Publications/Lav03cc.pdf

[21] Chow E, Hunkapiller T, Peterson J, Waterman MS. Biological information signal processor. In: Proc. of the Int'l Conf. on Application Specific Array Processor. 1991. 144−160. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=238887

[22] Roges T, Seeberg E. Six-Fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors. 2000,16:699−706. http://bioinformatics.oxfordjournals.org/content/16/8/699.abstract

[23] Farrar M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. Bioinformatics, 2007, 23(2):156−161.

[24] Martin WS, Delcuvillo JB, Useche FJ, Theobald KB, Gao GR. A multithreaded parallel implementation of a dynamic programming algorithm for sequence comparison. In: Proc. of the Pacific Symp. on Biocomputing. 2001. 311−322. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.6923&rep=rep1&type=pdf

[25] Yap TK, Frieder O, Martino RL. Parallel computation in biological sequence analysis. IEEE Trans. on Parallel and Distributed Systems, 1998,9(3):283−294.

[26] Muriki K, Underwood K, Sass R. RC-BLAST: Towards a portable, cost-effective open source hardware implementation. In: Proc. of the HiCOMB. 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5222005

[27] Lin H, Balaji P, Poole R, Sosa C, Ma X, Feng W. Massively parallel genomic sequence search on the blue Gene/P architecture. In: Proc. of the ACM/IEEE Conf. for High-Performance Computing, Networking, Storage and Analysis (SC). 2008. 1−11. http://en.cnki.com.cn/Article_en/CJFDTOTAL-JSJX200801014.htm

[28] Wu N, Wen M, He Y, Xun CQ, Ren J, Chai J, Zhang CY. MASA stream architecture and evaluating foir a fluid computing application. Chinese Journal of Computers, 2008,31(1):133−141 (in Chinese with English abstract).

附中文参考文献:

[16] 林伟,叶笑春,宋风龙,张浩.众核处理器中使用写掩码实现混合写回/写穿透策略.计算机学报,2008,31(11):1918−1928.

[18] 汪东,唐志敏.Smith-Waterman 算法在脉动阵列上的实现及分析.计算机学报,2004,27(1):12−20.

[19] 张法,乔香珍,刘志勇.基于 Smith-Waterman 算法的并行分而治之生物序列比对算法.中国科学(E 辑),2004,34(2):190−199.

[28] 伍楠,文梅,何义,荀长庆,任巨,柴俊,张春元.一种流处理器体系结构 MASA 及其在流体力学计算中的评测.计算机学报,2008,31(1):133−141.

叶笑春(1981－),男,江西万载人,博士,CCF 会员,主要研究领域为片上多核结构,并行计算,生物信息学.

范东睿(1979－),男,博士,副研究员,CCF 会员,主要研究领域为并行处理器设计.

林伟(1979－),男,博士,主要研究领域为计算机体系结构,处理器设计.

张浩(1980－),男,博士,助理研究员,CCF 会员,主要研究领域为片上众核结构,并行计算,处理器设计.