串行与并行数据结构与算法分析实验3

Five-Year-Plan

1、实验要求

实现n位二进制大整数的加法运算。输入a, b和输出s都是二进制位的串。要求算法的时间复杂度满足work=O(n)，span=O(log n)。

2、实验思路

2.1、加法计算

加法计算的难点在于保存进位信息，所以我们把结果分成两个部分："朴素和"和"进位"；其中"朴素和"表示1+1，1+0，0+0三种情况，"进位"表示对高位的影响。

不难发现每一位的相加都会决定下一位是否有进位，这种"错位传递"的特点和scan的特点十分相似，所以我们利用"朴素和"配合scan来得到"进位"的信息。

最终我们只要将"朴素和"与"进位"相加，并根据是否有溢出来决定是否要在高位补1。

2.2、减法计算

我们知道x>y，所以结果的长度必然不会超过x，这使得实现十分简单；我们只需要对y进行取反加一得到去除符号位的补码，然后两数相加，通过舍弃溢出位实现模运算，就可以得到正确的差。

PS：关于去除首位零的方法慢慢想......

2.3、乘法计算

乘法计算中我们可以使用分治法的思想将$A*B$看作$(p2^{n/2}+q)*(r2^{n/2}+s)$，从而分解成更小的乘法，如果我们直接分解为：$pr2^n+(ps+rq)2^{n/2}+qs$则递归式为：

$$W_{**}(n)=4W_{**}(n/2)+O(n), W_{**}(n)=O(n^2)$$

我们发现这个递归树的分支太多了，如果把4减少则可以得到更低的复杂度。如果我们把乘式分解为$pr2^n+[(p+q)*(r+s)-pr-qs]2^{n/2}+qs$，乘法就减少到了3次，那么复杂度降低：

$$W_{**}(n)=3W_{**}(n/2)+O(n), W_{**}(n)=O(n^{log_2 3})$$

3、回答问题

3.1、加法计算

Task 4.1 (35%). Implement the addition function

++ : bignum * bignum -> bignum

in the functor MkBigNumAdd in MkBigNumAdd.sml. For full credit, on input with *m* and *n* bits, yoursolution must have $O(m+n)$ work and $O(lg(m+n))$ span. Our solution has under 40 lines with comments.

```
(*
 * 大整数相加
 * 1）首先把两个bit串补为相同长度，方便后续计算；
 * 2）然后使用carry串储存“朴素和”：
 *      1+1用GEN表示；1+0用PROB表示；0+0用STOP表示
 * 3）然后在2的基础上使用scan对“朴素和”进行分析，使用的结合函数为：
 *      _, GEN => GEN      如果后一位是GEN则下一位必然会进位；
 *      _, STOP => STOP    如果后一位是STOP则下以为必然不会进位；
 *      some, PROP => some 如果后一位是PROB则它传递之前的进位情况；
 *    可以得出满足结合性；
 *    scan之后得到了每一位的进位情况：
 *      GEN表示进位，STOP表示不进位，PROP不会存在；
 *    多出的一位同时表示是否溢出；
 * 4）然后把“朴素和”和“进位信息”进行map2可以得到结果：
 * 5）最后看一下有没有溢出，如果就就在高位补充一个ONE就行了。
 *)
 fun x ++ y =
   case (length(x), length(y))
     of (0, 0) => empty()
      | (0, _) => y
      | (_, 0) => x
      | _ =>
   let
     (*1，高位补零使两串等长*)
     fun with0(a : bit seq, b : bit seq) =
       let val n = Int.max(length(a), length(b))
           val taila = tabulate (fn i => ZERO) (n - length a)
           val tailb = tabulate (fn i => ZERO) (n - length b)
       in (append(a, taila), append(b, tailb))
       end;
     (*2，得到“朴素和”*)
     fun getRawResult(x : bit seq, y : bit seq) =
       map2 (fn (i, j) => case (i, j)
                        of (ONE, ONE) => GEN
                         | (ZERO, ZERO) => STOP
                         | _ => PROP)
             x y;
     (*3，推导carry信息*)
     fun getCarryResult(x : carry seq) =
       scan (fn (i, j) => case (i, j)
                        of (_, GEN) => GEN
                         | (_, STOP) => STOP
                         | (some, PROP) => some)
             STOP x;
     (*4，直接得到结果*)
     fun getResult(x : carry seq, y : carry seq) =
       map2 (fn (i, j) => case (i, j)
```

```
                              of (PROP, GEN) => ZERO
                               | (PROP, STOP) => ONE
                               | (_, GEN) => ONE
                               | (_, STOP) => ZERO
                               | (_, _) => raise BugInGetResult)
                  x y;
      in
        let
          val (cx, cy) = with0(x, y)
          val rawResult = getRawResult(cx, cy)
          val (carryResult, high) = getCarryResult(rawResult)
          val result = getResult(rawResult, carryResult)
        in
          (*5，判断高位是否溢出*)
          if high = GEN then append(result, singleton ONE)
          else result
        end
      end;
```

3.2、减法计算

Task 4.2 (15%). Implement the subtraction function

-- : bignum * bignum -> bignum

in the functor MkBigNumSubtract in MkBigNumSubtract.sml, where *x* -- *y* computes the number

obtained by subtracting *y* from *x*. We will assume that *x*≥*y*; that is, the resulting number will always be non-negative. You should also assume for this problem that ++ has been implemented correctly. For full credit, if *x* has *n* bits, your solution must have *O*(*n*) work and *O*(lg*n*) span. Our solution has fewerthan 20 lines with comments.

```
(*
  * 大整数相减
  * 条件：x和y均为正数且x>y。
  * 1）给y高位补零使x和y等长，便于计算；
  * 2）对y进行取反加一，不考虑符号位；
  * 3）x与y的补码模相加，结果消除多余零；
  * 以上三个步骤分别使用下面的三个“过程”表示
  *)
  fun x -- y =
    if length y = 0 then x else
    if length y = 1 andalso nth y 0 = ZERO then x else
    let
      (*1、补0使之等长*)
      fun sameLen(x : bit seq, y : bit seq) =
        let val tail = tabulate (fn i => ZERO) (length(x)-length(y))
        in append(y, tail) end;
      (*2、取补码*)
      fun trueToComp(x : bit seq) : bit seq =
        (map (fn i => if i = ONE then ZERO else ONE) x) ++ singleton ONE;
```

```
      (*3、模相加并消零*)
      fun compSub (x : bit seq, cy: bit seq) : bit seq =
        take(x ++ cy, length(x));
      (*4、判断相减后是否为零*)
      fun isZero (x : bit seq) =
        (reduce (fn (i,j) => if i = ZERO andalso j = ZERO then ZERO else ONE)ZERO x) =
ZERO;
      val ans = compSub(x, trueToComp(sameLen(x, y)))
    in
      if isZero(ans) then empty() else ans
    end;
```

3.3、乘法计算

Task 4.3 (30%). Implement the function

** : bignum * bignum -> bignum

in MkBigNumMultiply.sml. For full credit, if the larger number has n bits, your solution must satisfy $W(n) = W(n/2) + O(n)$and have $O(lg^2 n)$ span. You should use the following function in the Primitives structure:

val par3 : (unit -> 'a) * (unit -> 'b) * (unit -> 'c) -> 'a * 'b * 'c

to indicate three-way parallelism in your implementation of **. You should assume for this problem that++ and -- have been implemented correctly, and meet their work and span requirements. Our solutionhas 40 lines with comments.

```
 (*
  * 大整数相乘
  * 使用分治法可以将n级别的乘法分成4个n/2级别的乘法，但是利用类似Strenssen?矩阵乘法的
  * 的技巧，可以用“便宜的”加减来换乘法，从而只需要3个n/2级别的乘法，work由O(n^2)下降
  * 为O(n^lg3)；span由于乘法可并行，保持为O(n)
  *)
  fun x ** y =
    case (length(x), length(y))
      of (0, _) => empty()
       | (_, 0) => empty()
       | (1, _) => if nth x 0 = ZERO then empty() else y
       | (_, 1) => if nth y 0 = ZERO then empty() else x
       | _ =>
    let
      (*补零为等长*)
      fun with0(a : bit seq, b : bit seq) =
        let
          val len = Int.max(length a, length b)
          val taila = tabulate (fn _ => ZERO) (len - length a)
          val tailb = tabulate (fn _ => ZERO) (len - length b)
        in (append(a, taila), append(b, tailb))
        end;
```

```
    val (nx, ny) = with0(x, y)
    (*取得半长*)
    val half = length(nx) div 2
    val q = take(nx, half)
    val p = drop(nx, half)
    val s = take(ny, half)
    val r = drop(ny, half)
    (*3次并行乘法*)
    val (p1, p2, p3) =
        par3(fn _ => p ** r,
             fn _ => q ** s,
             fn _ => (p ++ q) ** (r ++ s))
    val mm = tabulate (fn _ => ZERO) (half*2)
    val m  = tabulate (fn _ => ZERO) half
  in
    append(mm,p1) ++ append(m,p3 -- (p1 ++ p2)) ++ p2
  end;
```

3.4、迭代计算复杂度分析

Task 5.1 (15%). Determine the complexity of the following recurrences. Give tight *Θ*-bounds, and

justify your steps to argue that your bound is correct. Recall that *f*∈*Θ*(*g*) if and only if *f*∈*O*(*g*) and *g*∈*O*(*f*). You may use any method (brick method, tree method, or substitution) to show that your bound is correct, except that you must use the substitution method for problem 3.

$T(n)=3T(n/2)+\Theta(n)$

$T(n)=2T(n/4)+\Theta(\sqrt{n})$

$T(n)=4T(n/4)+\Theta(\sqrt{n})$(Prove by substitution)

1）$T(n)=3T(n/2)+\Theta(n)$

根据主方法，$\Theta(n^{log_2 3})>\Theta(n)$，叶节点掌控，故$T(n)=\Theta(n^{log_2 3})$

2）$T(n)=2T(n/4)+\Theta(\sqrt{n})$

根据主方法，$\Theta(n^{log_4 2})=\Theta(n^{1/2})=\Theta(\sqrt{n})$，平衡态，故$T(n)=\Theta(\sqrt{n}lg(n))$

3)$T(n)=4T(n/4)+\Theta(\sqrt{n})$

不妨令$\Theta(\sqrt{n})=c_0\sqrt{n}+d_0$

不妨假设若对$n=N/4$有$T(n)\leq c_1 n+c_2\sqrt{n}+d_1=\Theta(n),c_1>0$

则$T(N)\leq 4(c_1\frac{N}{4}+c_2\sqrt{\frac{N}{4}}+d_1)+c_0\sqrt{N}+d_0$

不妨取$c_2=-c_0,d_1=-\frac{d_0}{3}$

则$T(N)\leq c_1 N+c_2\sqrt{N}+d_1$也满足$T(n)\leq c_1 n+c_2\sqrt{n}+d_1$

由于显然存在$n_0>0\rightarrow c_1 n_0+c_2\sqrt{n_0}+d_1$

所以以上递归式成立

$T(n) = \Theta(n)$