Deepak Konidena
Apr 21, 2010

Fast Multiplication of Integers

Routines	Running times	Comments
Product1_Basic	O(mn) + c	Where m and n are the input sizes of operands in words and c is a constant time operation involving calloc() and memcpy() calls.
Karatsuba	$3*T_{((m+n)/2)} + 4*O(n) + 2*O((m+n)/2) + c$ Where m is the smaller number and c is for calloc() and memcpy() calls.	Fast Multiplication algorithm that recursively switches between itself and Product1_Basic.
Difference	O(n) Where n is the larger number.	Calculates the difference between two input operands for numbers whose base is larger than the word size.
Addition1_4	O(n) Where n is the larger number	Calculates the sum of the input operands for very large numbers. Typically ones whose base is larger than the word size.
toBinary		Used for printing unsigned integers in binary. Mainly used for debugging purposes.
toBinary64		Used for printing unsigned long long integers in binary. Again used for debugging purposes.
Product32		Main program that calls either Karatsuba or Product1_Basic based on the input size.

The table laid out below describes the runtime performance of Product1_8_Basic when compared to that of GMP.

Size of operands, Number of multiplications	GMP (runtime in sec)	Product1_8_Basic (runtime in sec)	(Product1_8_Basic) / GMP
10 , 100	0.000039	0.000095	2.462
100, 100	0.000138	0.000244	1.766
1000, 100	0.000246	0.015255	61.93
10000, 100	0.02516	0.568370	22.57
100000, 10	0.037210	5.951381	159.938
1000000, 1	0.06696	57.86420	864.334

Note : The truncation bit is set while performing the multiplications and all the results are correct. The times shown here are based on one random run.

Code

```
/*
 * scaffold32.c
 *
 *  Created on: Mar 28, 2010
 *      Author: deepakkoni
 */

#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>
#include <time.h>
#include "scaffold32.h"
#include <math.h>
#include <string.h>
#include <limits.h>


char* toBinary(int32 i, char *str){

      unsigned int num = i;
      strcpy(str, "");

      while(num){
        if( 0x0001 & num){
               strcat(str,"1");
               num = num >> 1;
        }
        else{
               strcat(str,"0");
               num = num >> 1;
        }
      }
      return str;
}

char* toBinary64(int64 i, char *str){

      unsigned long long int num = i;
      strcpy(str, "");

      while(num){
        if( 0x0001 & num){
               strcat(str,"1");
               num = num >> 1;
        }
        else{
               strcat(str,"0");
               num = num >> 1;
        }
      }
      return str;
}

int32** Product1_8_Basic(int32** a, int32** b, int32** c, int32 wa, int32 wb){

        int32 i, j, carry, flag= 0;
        int64 base = (int64)1<<32, p;
```

```
        if(wa > wb)
               flag = 1;
        else if(wa < wb)
               flag = 2;
        else if (wa == wb)
               flag = 0;


          if(flag == 0){

              //zeroing out the remaining digits
              for(i = 0; i <      wa ; i++ )
                     *c[i] = 0;

              //Actual 1.8 implementation
              for(i = 0; i< wa; i++){
                     carry = 0;
                     for(j = 0; j < wa; j++){

                            p = (int64)(*a[i])*(*b[j]) + *c[i + j] + carry;
                            *c[i+j] = (int32)(p & (0xffffffff));
                            carry = (int32)(p / base);
                     }
                     *c[i+wa] = carry;
              }
           }

           if(flag == 1){
                //zeroing out the remaining digits
                for(i = 0; i < wa ; i++ )
                   *c[i] = 0;

                //Actual 1.8 implementation
                for(i = 0; i< wa; i++){
                      carry = 0;
                   for(j = 0; j < wb; j++){
                      p = (int64)(*a[i])*(*b[j]) + *c[i + j] + carry;
                      *c[i+j] = (int32)(p & (0xffffffff));
                      carry = (int32)(p / base);
                   }
                   *c[i+wb] = carry;
                }
         }

           if(flag == 2){
                //zeroing out the remaining digits
                for(i = 0; i < wb ; i++ )
                   *c[i] = 0;
                //Actual 1.8 implementation
                for(i = 0; i< wb; i++){
                      carry = 0;
                   for(j = 0; j < wa; j++){
                      p = (int64)(*b[i])*(*a[j]) + *c[i + j] + carry;
                      *c[i+j] = (int32)(p & (0xffffffff));
                      carry = (int32)(p / base);
                   }
                   *c[i+wa] = carry;
                }
         }
              return &c[0];
}
```

```
void Karatsuba(int32** a, int32** b, int32** res, int32 dig, int32 wa, int32
wb){

      //a is always either larger or equal to b
      // u1, u2 point to larger number and v1,v2 point to smaller number
      int32 i;
      int32 **u1, **v2, **u2;

      if(dig == 1){
        Product1_8_Basic(a,b,&res[5*(wa + wb)],wa,wb);
        for(i = 0; i < (wa+wb); i++)
               printf(" %x",*res[5*(wa + wb) + i]);
      }


      else{
        if(wa == wb){

               int32  **v1;

               if((u1 = (int32 **)calloc((wa - dig),sizeof(int32*)))== NULL)
                      fprintf(stderr, "\n Memory cannot be allocated to u1\n");
               if((u2 = (int32 **)calloc(dig,sizeof(int32*)))== NULL)
                      fprintf(stderr, "\n Memory cannot be allocated to u2\n");
               if((v2 = (int32 **)calloc(dig,sizeof(int32*)))== NULL)
                      fprintf(stderr, "\n Memory cannot be allocated to v1\n");
               if((v1 = (int32 **)calloc((wb - dig),sizeof(int32*)))== NULL)
                      fprintf(stderr, "\n Memory cannot be allocated to v2\n");

                 //Allocating space for individual pointers
               for(i = 0; i < (wa - dig); i++)
                  if((u1[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
                         fprintf(stderr,"\nMemory cannot be allocated\n");
               for(i = 0; i < dig; i++)
                  if((u2[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
                         fprintf(stderr,"\nMemory cannot be allocated\n");
               for(i = 0; i < dig; i++)
                  if((v2[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
                         fprintf(stderr,"\nMemory cannot be allocated\n");
               for(i = 0; i < (wb - dig); i++)
                  if((v1[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
                         fprintf(stderr,"\nMemory cannot be allocated\n");

              //populating u1, u2, v1, v2
             //copying values
             for(i = 0; i < dig; i++)
                 memcpy(u2[i], a[i], sizeof(int32));
             for(i = 0; i < (wa - dig); i++)
                memcpy(u1[i], a[i + dig], sizeof(int32));
             for(i = 0; i < dig; i++)
                 memcpy(v2[i], b[i], sizeof(int32));
             for(i = 0; i < (wb -dig); i++)
                 memcpy(v1[i], b[i + dig], sizeof(int32));

             //printing u1, u2, v1, v2
             for(i = 0; i < (wa - dig); i++)
                printf("\n u1[%u] = %x" ,i, *u1[i]);
             for(i = 0; i < dig; i++)
                printf("\n u2[%u] = %x" ,i, *u2[i]);
             for(i = 0; i < (wb - dig); i++)
```

```
            printf("\n v1[%u] = %x" ,i, *v1[i]);
          for(i = 0; i < dig; i++)
            printf("\n v2[%u] = %x" ,i, *v2[i]);

          //Calculating the sum (U1 + U2)
            Addition1_4(u1, u2, (wa - dig), dig, &res[2*(wa + wb)]);
            Addition1_4(v1, v2, (wb - dig), dig, &res[3*(wa + wb)]);

            //Printing (U1+U2) and (V1+V2)
            for(i = 0; i < (wa - dig); i++)
                  printf("\n (u1+u2) = %x", *res[2*(wa + wb) + i]);
            for(i = 0; i < (wb - dig); i++)
                  printf("\n (v1+v2) = %x", *res[3*(wa + wb) + i]);

            //product of T1,T2 & U1,V1 and U2,V2
            Product1_8_Basic(&res[2*(wa + wb)], &res[3*(wa + wb)], &res[4*(wa
+ wb)], (wa - dig), (wb - dig) );
            Product1_8_Basic(&u1[0], &v1[0], &res[0], (wa - dig), (wb - dig)
);
            Product1_8_Basic(&u2[0], &v2[0], &res[(wa+wb)], dig, dig );

            for(i = 0; i < (wa + wb - 2*dig); i++)
                  printf("\n (u1 + u2)*(v1 + v2) = %x" , *res[4*(wa + wb) +
i]);
            for(i = 0; i < (wa + wb - 2*dig); i++)
                printf("\n (u1)*(v1) = %x" , *res[i]);
            for(i = 0; i < (2*dig); i++)
                printf("\n (u2)*(v2) = %x" , *res[(wa + wb) + i]);

            //Calculating the difference of W3=(T1*T2), W2 = (U1*V1) and W4 =
(U2*V2)
            Difference(&res[4*(wa+wb)], &res[0], (wa + wb - 2*dig), (wa + wb -
2*dig),&res[6*(wa + wb)]);
            Difference(&res[6*(wa+wb)], &res[(wa + wb)], (wa + wb - 2*dig),
(2*dig),&res[7*(wa + wb)]);

            for(i = 0; i < (wa + wb - 2*dig); i++)
                  printf("\n (u1+ u2)*(v1 + v2) - u1*v1 = %x" ,*res[6*(wa +
wb) + i]);
            for(i = 0; i < (wa + wb - 2*dig); i++)
                  printf("\n (u1+ u2)*(v1 + v2) - u1*v1 - u2*v2= %x" ,
*res[7*(wa + wb) + i]);

            // freeing the pointers
                for(i = 0; i < (wb - dig); i++)
                    free(v1[i]);

                free(v1);
      }



      else{


            if((u1 = (int32 **)calloc((wa - dig),sizeof(int32*)))== NULL)
                  fprintf(stderr, "\n Memory cannot be allocated to u1\n");
            if((u2 = (int32 **)calloc(dig,sizeof(int32*)))== NULL)
                  fprintf(stderr, "\n Memory cannot be allocated to u2\n");
            if((v2 = (int32 **)calloc(dig,sizeof(int32*)))== NULL)
                  fprintf(stderr, "\n Memory cannot be allocated to v1\n");
```

```
              //Allocating space for individual pointers
             for(i = 0; i < (wa - dig); i++)
                if((u1[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
                       fprintf(stderr,"\nMemory cannot be allocated\n");
             for(i = 0; i < dig; i++)
                if((u2[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
                       fprintf(stderr,"\nMemory cannot be allocated\n");
             for(i = 0; i < dig; i++)
                if((v2[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
                       fprintf(stderr,"\nMemory cannot be allocated\n");


             //populating u1, u2, v1, v2
           //copying values
           for(i = 0; i < dig; i++)
               memcpy(u2[i], a[i], sizeof(int32));
           for(i = 0; i < (wa - dig); i++)
             memcpy(u1[i], a[i + dig], sizeof(int32));
           for(i = 0; i < dig; i++)
               memcpy(v2[i], b[i], sizeof(int32));


           //printing u1, u2, v1, v2
           for(i = 0; i < (wa - dig); i++)
             printf("\n u1[%u] = %x" ,i, *u1[i]);
           for(i = 0; i < dig; i++)
             printf("\n u2[%u] = %x" ,i, *u2[i]);
           for(i = 0; i < dig; i++)
             printf("\n v2[%u] = %x" ,i, *v2[i]);

           //Calculating the sum (U1 + U2)
           if((wa - dig) > dig ){
             Addition1_4(u1, u2, (wa - dig), dig, &res[2*(wa + wb)]);
             for(i = 0; i < dig; i++)
                    memcpy(res[3*(wa + wb) + i], v2[i], sizeof(int32));

             //Printing (U1+U2) and (V1+V2)
             for(i = 0; i < (wa - dig); i++)
                    printf("\n (u1+u2) = %x", *res[2*(wa + wb) + i]);
             for(i = 0; i < dig; i++)
                    printf("\n (v1+v2) = %x", *res[3*(wa + wb) + i]);

             //product of T1,T2 & U1,V1 and U2,V2
             Product1_8_Basic(&res[2*(wa + wb)], &res[3*(wa + wb)], &res[4*(wa
+ wb)], (wa - dig), dig );
             Product1_8_Basic(&u2[0], &v2[0], &res[(wa+wb)], dig, dig );

             for(i = 0; i < (wa); i++)
                    printf("\n (u1 + u2)*(v1 + v2) = %x" , *res[4*(wa + wb) +
i]);
             for(i = 0; i < (2*dig); i++)
                    printf("\n (u2)*(v2) = %x" , *res[(wa + wb) + i]);

             //Calculating the difference of W3=(T1*T2), W2 = (U1*V1) and W4 =
(U2*V2)
                    Difference(&res[4*(wa+wb)], &res[(wa + wb)], (wa),
(2*dig),&res[4*(wa + wb)]);

                    for(i = 0; i < (wa); i++)
```

```
                        printf("\n (u1+ u2)*(v1 + v2) - u1*v1 - u2*v2= %x" ,
*res[4*(wa + wb) + i]);

           }
           else{
              Addition1_4(u2, u1, dig, (wa -dig), &res[2*(wa + wb)]);

              for(i = 0; i < dig; i++)
                     memcpy(res[3*(wa + wb) + i], v2[i], sizeof(int32));

              //Printing (U1+U2) and (V1+V2)
              for(i = 0; i < dig; i++)
                     printf("\n (u1+u2) = %x", *res[2*(wa + wb) + i]);

              for(i = 0; i < dig; i++)
                     printf("\n (v1+v2) = %x", *res[3*(wa + wb) + i]);

              //product of T1,T2 & U1,V1 and U2,V2
              Product1_8_Basic(&res[2*(wa + wb)], &res[3*(wa + wb)], &res[4*(wa
+ wb)], dig, dig );
              Product1_8_Basic(&u2[0], &v2[0], &res[(wa+wb)], dig, dig );

              for(i = 0; i < 2*dig; i++)
                     printf("\n (u1 + u2)*(v1 + v2) = %x" , *res[4*(wa + wb) +
i]);
              for(i = 0; i < 2*dig; i++)
                     printf("\n (u2)*(v2) = %x" , *res[(wa + wb) + i]);

              //Calculating the difference of W3=(T1*T2), W2 = (U1*V1) and W4 =
(U2*V2)
                     Difference(&res[4*(wa+wb)], &res[(wa + wb)], 2*dig,
(2*dig),&res[6*(wa + wb)]);

                     for(i = 0; i < 2*dig; i++)
                            printf("\n (u1+ u2)*(v1 + v2) - u1*v1 - u2*v2= %x" ,
*res[6*(wa + wb) + i]);
           }


       }

     // freeing the pointers
         for(i = 0; i < (wa - dig); i++)
             free(u1[i]);
         for(i = 0; i < (dig); i++)
             free(u2[i]);
         for(i = 0; i < (dig); i++)
             free(v2[i]);
         free(u1);
         free(u2);
         free(v2);
         //free(str);
     }

}

void Addition1_4(int32** x, int32** y, int32 wx, int32 wy, int32** sum){

     int32 i, carry = 0;
     int64 base = (int64)1 << 32, s = 0;
```

```
      for(i = 0; i < wy; i++){
        s = (int64)(*x[i]) + *y[i] + carry;
        *sum[i] = (int32)(s % base);
        carry = (int32)(s/base);
      }

      for(i = wy; i < wx; i++){
        s = (int64)(*x[i]) + carry;
        *sum[i] = (int32)(s % base);
        carry = (int32)(s/base);
      }
}

void Difference(int32** x, int32** y, int32 wx, int32 wy, int32** diff){

      int32 i, borrow = 0;
      int64 base = (int64)1 << 32, s = 0;


      for(i =0; i < wy; i++){
        s = (int64)(*x[i]) - *y[i] - borrow;
        *diff[i] = (int32)(s % base);
        //printf("\n%x - %x - %x = %x ", *x[i], *y[i], borrow, *diff[i]);
        borrow = (int32)(s/base);
        if(borrow != 0)
              borrow = 1;
      }

      for(i = wy; i < wx; i++){
        s = (int64)(*x[i]) - borrow;
        *diff[i] = (int32)(s % base);
        borrow = (int32)(s/base);
        if(borrow != 0)
              borrow = 1;
      }
}

/* You are suppose to change the routine Product32 here to your own routine
 * The mpz calls in the scaffolded Product32 below are the normal GMP function
 * calls and should be neglected. By casting the void pointers as normal
unsigned
 * integers, you should be able to access the data values as normal 4 bytes
words.
 */


/* wa is word length of a, ba is bit length of a */
void Product32(void *a, void *b, void *c, unsigned int wa, unsigned int ba,
unsigned int wb, unsigned int bb, unsigned int *wc, unsigned int *bc){

      int32 SMALLER_WORDSIZE = 0;
      int32 i = 0;

     /* Cast a and b into short integers of size 32 bits */
     int32 *int_a = (int32 *) a;
     int32 *int_b = (int32 *) b;
     int32 *int_c = (int32 *) c;

     // Finding the smaller of wa and wb
     if(wa == wb)
      SMALLER_WORDSIZE = wa/2;
```

```
    else
     SMALLER_WORDSIZE = (wa < wb) ? wa : wb;

    *wc = wa + wb;

  //  printf("\n wa = %u, wb = %u \n", wa, wb);


    //creating space for and copying a and b, creating space for c
    int32** int32_a = (int32 **)calloc(wa, sizeof(int32 *));
    if(int32_a == NULL)
     fprintf(stderr,"\nMemory cannot be allocated\n");
    int32** int32_b = (int32 **)calloc(wb, sizeof(int32 *));
    if(int32_b == NULL)
        fprintf(stderr,"\nMemory cannot be allocated\n");
    int32** int32_c = (int32 **)calloc(*wc, sizeof(int32 *));
    if(int32_c == NULL)
        fprintf(stderr,"\nMemory cannot be allocated\n");


    //Allocating space for individual pointers
    for(i = 0; i < wa; i++)
     if((int32_a[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
        fprintf(stderr,"\nMemory cannot be allocated\n");
    for(i = 0; i < wb; i++)
     if((int32_b[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
        fprintf(stderr,"\nMemory cannot be allocated\n");
    for(i = 0; i < *wc; i++)
     if((int32_c[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
        fprintf(stderr,"\nMemory cannot be allocated\n");

  /*  //Allocating space for recursive workspace
    int32** result = (int32 **)calloc(8*(wa + wb), sizeof(int32 *));
        if(result == NULL)
             fprintf(stderr,"\nMemory cannot be allocated\n");

    for(i = 0; i < 8*(wa + wb); i++)
     if((result[i] = (int32 *)calloc(1, sizeof(int32))) == NULL)
            fprintf(stderr,"\nMemory cannot be allocated\n");
*/
    //copying values
    for(i = 0; i < wa; i++)
        memcpy(int32_a[i], &int_a[i], sizeof(int32));
    for(i = 0; i < wb; i++)
     memcpy(int32_b[i], &int_b[i], sizeof(int32));

 /*   printf("\n");
    //printing the values
    for(i = 0; i < wa; i++)
         printf(" a[%u] = %x" ,i, *int32_a[i]);
    printf("\n");
     for(i = 0; i < wb; i++)
         printf(" b[%u] = %x" ,i, *int32_b[i]);
*/
  /* if(wa > wb || wa == wb)
     //karatsuba
      Karatsuba(int32_a, int32_b, result , SMALLER_WORDSIZE, wa, wb);
     else if(wa < wb)
      Karatsuba(int32_b, int32_a, result , SMALLER_WORDSIZE, wb, wa);
     */
```

```
    /*
    printf("\nThe value of 1<<32 is %s", toBinary64((int64)1<<32 , str));
    printf("\nThe size of unsigned short int is %u bytes", sizeof(int16));
    printf("\nThe size of unsigned int is %u bytes", sizeof(int32));
    printf("\nThe size of unsigned long int is %u bytes", sizeof(int32));
    printf("\nThe size of unsigned long long int is %u bytes", sizeof(int64));
     */
     Product1_8_Basic(int32_a, int32_b, int32_c, wa, wb);

     for(i = 0; i < *wc; i++)
      memcpy(&int_c[i],int32_c[i], sizeof(int32));



     // freeing the pointers
         for(i = 0; i < wa; i++)
             free(int32_a[i]);
         for(i = 0; i < wb; i++)
             free(int32_b[i]);
         for(i = 0; i < *wc; i++)
             free(int32_c[i]);

      /*   for(i = 0; i < 8*(wa + wb); i++)
             free(result[i]);
*/
         free(int32_a);
         free(int32_b);
         free(int32_c);
  //     free(result);

}
```