Advanced practical **Programming** for Scientists

Thorsten Koch

Zuse Institute Berlin

TU Berlin

WS2014/15

Variants

opt33 Intel® Xeon ® E3-1245 v3 @ 3.40GHz (Haswell)

opt17 Intel ® Xeon ® E3-1280 @ 3.50GHz (Ivy Bridge)

griesu Intel ® Core™ i7-640M @ 2.80GHz (Westmere)

gcc 4.8.2

clang 3.4

icc 15.0.1 (opt17,opt33) -O3 –march=native

gcc 4.7.2 (griesu)

5 modes: int longlong float double longdouble

4 algos : intrinsic debruijn bitmask loop

3 passes : onepass unroll normal

= 60 variants per compiler / coverage: -DNDEBUG –DNO_MSHELL

Results opt17/gcc

int	intrinsic	unroll	3.069
int	debruijn	unroll	3.255
float	intrinsic	unroll	3.405
int	intrinsic	normal	3.446
int	loop	unroll	3.526
int	bitmask	unroll	3.627
int	debruijn	normal	3.661
float	debruijn	unroll	3.734
float	loop	unroll	3.776
double	intrinsic	unroll	3.858
double	debruijn	unroll	3.913
float	intrinsic	normal	3.917
float	bitmask	unroll	3.948
float	debruijn	normal	3.970
double	intrinsic	normal	4.007
double	bitmask	unroll	4.134
int	loop	normal	4.139
float	loop	normal	4.164
int	bitmask	normal	4.291
double	loop	unroll	4.321
double	debruijn	normal	4.345
float	bitmask	normal	4.383
longlong	intrinsic	unroll	4.383
double	loop	normal	4.416
longlong	loop	unroll	4.602
longlong	debruijn	unroll	4.619
double	bitmask	normal	4.670
longlong	bitmask	unroll	4.971
longlong	debruijn	normal	5.789
longlong	loop	normal	5.822
longlong	intrinsic	normal	5.856
double	intrinsic	onepass	6.474
double	debruijn	onepass	6.685
double	loop	onepass	6.688
longlong	bitmask	normal	6.859
double	bitmask	onepass	6.878
int	intrinsic	onepass	7.092
int	loop	onepass	7.120
int	debruijn	onepass	7.301
int	bitmask	onepass	7.579
float	loop	onepass	7.947
float	intrinsic	onepass	8.065
float	debruijn	onepass	8.349
float	bitmask	onepass	8.451
longlong	debruijn	onepass	10.775
longlong	loop	onepass	10.779
longlong	intrinsic	onepass	11.021
longlong	bitmask	onepass	11.026
longdouble	intrinsic	unroll	16.618
longdouble	debruijn	unroll	16.700
longdouble	bitmask	unroll	16.856
longdouble	loop	unroll	17.080
longdouble	intrinsic	normal	18.102
longdouble	debruijn	normal	18.201
longdouble	loop	normal	18.314
longdouble	bitmask	normal	18.412
longdouble	intrinsic	onepass	22.760
longdouble	debruijn	onepass	22.879
longdouble	loop	onepass	23.253
longdouble	bitmask	onepass	23.318

Code

```
#Includes in .h files
Use const in parameters
extern TYPE* bp_getRhs(BinaryProgram* bp);
#if 0 / #endif

#ifndef _TYPE_BP_H_
#define _TYPE_BP_H_
#include "struct_bp.h"
typedef struct BP BinaryProgram;
#endif /* _TYPE_BP_H_ */

Allocate.c not needed if mshell is there

Use doxygen
/*
 * creates a new binary program
 */
BinaryProgram* bp_new(
```

```
   if (max <= rhs)
   {
#ifdef MORE_DEBUG
      printf("Constraint is redundant.\n");
#endif
      bp->redundant++;
      return BP_OKAY;
   }

   /* constraint is infeasible, when minimal activity is greater than rhight-hand side
    * => return infeasible
    */
   if (min > rhs)
      return BP_INFEASIBLE;
```

Many returns from a function.

```
for (; j < bp->n; j++)
{
   /* nothing to do, when both coefficients are zero */
   if (coefs[j] == 0 && bp->coefs[j] == 0)
      continue;
   /* break, when one coefficient is zero and the other is negative */
   else if ((coefs[j] == 0 && bp->coefs[j] < 0) || (coefs[j] < 0 && bp->coefs[j] == 0))
      break;
   /* only stored constraint can dominates new one, when coefficient of new one is zero */
   else if (coefs[j] == 0 && bp->coefs[j] > 0)
   {
      if (dom == -1 || quot != 0)
         break;
      dom = 1;
   }
   /* only new constraint can dominates stored one, when coefficient of stored one is zero */
   else if (coefs[j] > 0 && bp->coefs[j] == 0)
   {
      if (dom == 1 || quot != 0)
         break;
      dom = -1;
   }
   else
   {
      assert(coefs[j] != 0);
      assert(bp->coefs[j] != 0);
      double q = coefs[j] / bp->coefs[i*bp->n+j];

      /* compute quotient for positive coefficients */
      if (coefs[j] > 0 && bp->coefs[i*bp->n+j] > 0)
      {
         if (quot != 0)
            break;
```

```
#ifdef DEBUG
#ifndef MORE_DEBUG
      /* print solution */
      if (valid)
      {
#endif
         for (int k = 0; k < bp->n; k++)
         {
            fprintf(fp, "%i", values[k]);
            if (k < bp->n-1)
            {
               fprintf(fp, " ");
            }
            else
            {
#ifndef MORE_DEBUG
               fprintf(fp, "\n");
#else
               fprintf(fp, ": %i\n", valid);
#endif
            }
         }
#ifndef MORE_DEBUG
      }
#endif
#endif
   }
```

Do not use #if / #endif

```
extern lp_free(LinearProgram* lp);
bool lp_is_valid(LinearProgram* lp);
extern uint64_t get_bin_solutions_lp(LinearProgram* lp);

const  / no extern

void lp_free(LinearProgram* lp) {
    assert(lp_is_valid(lp));
    int i;
    for (i = 0; i < lp->rows; i++) {
        deallocate(lp->matrix[i]);
    }

Either use c89 or c99
```

```
#include <assert.h>
#include <limits.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "allocate.h"
#include "linear_program.h"

#define GET_SEC(a, b) ((double) (b - a) / (double)CLOCKS_PER_SEC)

struct linear_program {
    int rows;
    int cols;
    num_t** matrix;
    num_t* vector;
    int* constraint_types;
};

bool lp_is_valid(LinearProgram* lp) {
    if (NULL == lp) {
```

Documentation!

```
int rows = 0;
int cols;
char* end_ptr = NULL;
LinearProgram* lp = NULL;
```

```
int           rows    = 0;
int           cols;
char*         end_ptr = NULL;
LinearProgram* lp     = NULL;
```

```
struct linearProgram* createLPFromFile(const char*);
int validateNumber(char*);
int fillRow(num*, int, char*, char*);
int parseDecimal(char*);

var names, typedef


if (!(col > 0 && col <=32 && row > 0 && row <= 32)) {

if (col < 0 || col > 32 || row < 0 || row > 32) {
```

```
   if (argc < 2 || strlen(argv[1]) <= 0)
    {
    fprintf(stderr, "usage: %s filename\n", argv[0]);
    return EXIT_FAILURE;
    }
// indentation
```

```
     printf("Starting new enumeration...\n\n");
     if (IS_DOUBLE) {
         printf("\nall values considered as 'double'...\n\n");
     }
      else {
         printf("all values considered as 'int'\n");
      }
```

```
printf("all values are considered as %s\n", IS_DOUBLE ? “int“ : “double“);

printf("Laufzeit der LP-Erstellung: %.8f Sekunden\n",(float)(create_end-
create_start) / CLOCKS_PER_SEC);
```

```
LinearProgram* lp = allocate(1, sizeof(*lp));
lp->var = var;
lp->constr = 0;
/* since == constraints are replaced by two <= constraints,
   we have to reserve space for twice the given number of constraints */
lp->max_constr = 2 * constr;
lp->matrix = allocate(lp->max_constr, sizeof(*lp->matrix));
int i;
for (i = 0; i < lp->max_constr; ++i)
   lp->matrix[i] = allocate(var, sizeof(*lp->matrix[0]));
lp->rhs = allocate(lp->max_constr, sizeof(*lp->rhs));
```

```
LinearProgram* lp = allocate(1, sizeof(*lp));
lp->var           = var;
lp->constr        = 0;
lp->max_constr    = 2 * constr; // since == constraints are replaced by two …
lp->matrix        = allocate(lp->max_constr, sizeof(*lp->matrix));
lp->rhs           = allocate(lp->max_constr, sizeof(*lp->rhs));

for (int i = 0; i < lp->max_constr; i++)
   lp->matrix[i]  = allocate(var, sizeof(*lp->matrix[0]));
```

```
int gray_get_sign(
      unsigned long index, ///< the index of the new code word
      int changed_bit ///< the bit that changes
                  )
{
   assert(index >= 1);
   assert(changed_bit == gray_get_changed_bit(index));

   if(((1ul << (changed_bit + 1)) & index) == 0ul)
      return 1;
   else
      return -1;
}
return (((1ul << (changed_bit + 1)) & index) ? -1 : 1;

* Calculates the bit that shifts when going to the code word with the given index.
*/
int gray_get_changed_bit(
      unsigned long index ///< the index of the new code word
       )
{
   assert(index >= 1);

   return __builtin_ffsl(index) - 1;    // Portability
}
```

Reserved names

```
typedef struct
{
  int rows;
  int cols;
  DBLINT* matrix;
  cmp_t* cmp;
  DBLINT* rhs; // righthand-side
} __constraints_intern_t;
```

In addition to the names documented in this manual, reserved names include all external identifiers (global functions and variables) that begin with an underscore (‘_’) and all identifiers regardless of use that begin with either two underscores or an underscore followed by a capital letter are reserved names. This is so that the library and header files can define functions, variables, and macros for internal purposes without risk of conflict with names in user programs.

Names that end with ‘_t’ are reserved for additional type names.

```
// TODO: Support the other relation symbols, too.
 if((LESSEQ != cs.cmp[r1]) || (LESSEQ != cs.cmp[r2]))
   return false;
```

```
 int row=0;
 for(row=0; row<cs.rows; row++)
   if(valid)
     valid = ((LESSEQ == cs.cmp[row]) || (GREATEQ == cs.cmp[row]) || …
   else
     break;
```

```
   for(int row = 0; row < cs.rows; row++)
   {
      if (!valid)
         break;
      valid = ((LESSEQ == cs.cmp[row]) || (GREATEQ == cs.cmp[row]) || …
   }
```

```
   for(int row = 0; valid && row < cs.rows; row++)
      valid = ((LESSEQ == cs.cmp[row]) || (GREATEQ == cs.cmp[row]) || …
```

```
int r;
switch(r=read_constArray_from_file(argv[1], &c)){
case -1:
    return EXIT_FAILURE;
```

```
int r = read_constArray_from_file(argv[1], &c);

switch(r)
{
case -1:
    return EXIT_FAILURE;
```

```
if (0 != strcmp(debug, "\0")) {
    return -1;
} else {
    *ptrToNumb = value;
    return 1;
}
```

```
if (strcmp(debug, ""))
    return -1;

*ptrToNumb = value;

return 1;
```

```
if (debug[0] != '\0')     /// better ???
    return FAIL_XXX;
```

```
char* string;
string = getBinaryFromInt(binary, dimension);
```

```
if (numbsEqual(coeffI, 0)) {
     if (numbsEqual(coeffJ, 0)) {
         continue;
     } else {
         scalable = 0;
         continue;
     }
 }
```

```
if (numbsEqual(coeffI, 0))
{
     if (numbsEqual(coeffJ, 0))
         continue;

      scalable = 0;
     continue;
}
```

getopt

The nice thing about standards is that you have so many to choose from.

-- Andrew Tannenbaum

man page says POSIX.2 getopt from unistd.h

-std=c99 -> pure ansi, no posix

-std=gnu99 -> gives all gnu extentions

getopt.h is a glibc extention including getopt_long()

-D_POSIX_C_SOURCE=200809L -D_XOPEN_SOURCE=700

strdup can have similar issues.

Getting warnings away

```
#define MACROSTR(s) #s

#define ADD_TO_ADDR(t, p, a)    \
  _Pragma("clang diagnostic push") \
  _Pragma(MACROSTR(clang diagnostic ignored "-Wcast-align"))  \
  ((t)(((char*)(p)) + (a)))         \
  _Pragma("clang diagnostic pop")

#define CLIENT_2_HDR(a) ADD_TO_ADDR(MHDR*, a, -RESERVE_SIZE)
#define HDR_2_CLIENT(a) ADD_TO_ADDR(void*, a, RESERVE_SIZE)
```

See what is happening

```
//#define USE_INTRINSIC  1
//#define USE_DEBRUIJN   1
//#define USE_BITMASK    1
//#define USE_LOOP       1  // (default)

#if !defined(USE_INTRINSIC) && !defined(USE_DEBRUIJN) &&
      !defined(USE_BITMASK) && !defined(USE_LOOP)
#error "One of USE_INTRINSIC, USE_DEBRUIJN, USE_BITMASK, USE_LOOP has
        to be defined"
#endif
```

flexelint

```
//lint -sem( bip_read, nulterm(1), @p == 1)
extern BIP* bip_read(const char* filename);


--- Module:  src/ex6.c (C)
  if (NULL == bip)
        ^
src/ex6.c:125: Info 774: Boolean within 'if' always evaluates to False
[Reference: file src/ex6.c: lines 34, 123, 125; file
  src/bip.h: line 34]
src/ex6.c:34: Info 831: Reference cited in prior message
src/ex6.c:123: Info 831: Reference cited in prior message
src/ex6.c:125: Info 831: Reference cited in prior message
src/bip.h:34: Info 831: Reference cited in prior message
```

Exercise

Rewrite the program from exercise 1 in Ada.