Optimización

José Galaviz

¿cuando se optimiza?

- Diseño: haciéndolo simple.
- Implementación: preservándolo simple.
- Depuración: simplificando lo que era suficientemente complicado para estar mal.
- Si aún así hace falta... entonces habrá que hacer algo más específico.

Prioridades

1. Que el programa haga lo que debe.
2. Que el programa sea robusto.
3. Que el programa facilite su mantenimiento (entre lo que se incluye ser legible).
4. Que el programa sea eficiente.

Quien optimiza

- El programador al elaborar el programa.
  - Escogiendo la estructura de datos adecuada.
  - Con los algoritmos adecuados.
  - Sin redundancias en el código.
- El compilador.
  - En diversos niveles.
- El programador.
  - Con base en lo observado en la ejecución.

Optimizaciones de gcc

Bandera	Descripción
-O0	Compilación simple. No se hacen optimizaciones. El código obtenido preserva toda la información para depuración
-O1	Transformaciones que preservan el orden del código. No desaparecen variables.

Optimizaciones de gcc

Bandera	Descripción
-O2	Transformaciones más agresivas que dan lugar a código más rápido, aunque sin incrementar su tamaño. Algunas variables y funciones pueden desaparecer.
-O3	Se hacen optimizaciones mayores, reordenamiento de código, el resultado puede o no ser más rápido que el original y frecuentemente se hace más grande el ejecutable.

Generalidades

- Normalmente no se utilizan niveles mayores a 2. Es el nivel de distribución de los programas de GNU.
- No se compromete seriamente la depurabilidad del programa.
- No crece el tamaño del ejecutable.

Nivel 1

- Se difiere el pop de las llamadas a subrutinas (defer pop).
- Dado un salto condicional se verifican las condiciones de los saltos en los que se caería (si hay, thread jumps).
- El alojamiento en registros se propaga para no tener que volver a asignar (prop registers).
- Se tratan de adivinar los saltos (guess branch).

Nivel 2

- Se linean funciones, datos, saltos y etiquetas.
- Se eliminan subexpresiones comunes (CSE).
- Se presupone que no existirán dos referencias de diferente tipo a un mismo lugar de memoria (strict aliases).
- Se elimina el código de verificación de referencias nulas.
- Se reacomodan bloques de código.

Nivel 3

- Se renombran registros.
- Se reemplazan llamadas a funciones por el código de las mismas (inline functions).

Más

http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Herramientas de optimización

- Para optimizar el código más allá de lo evidente habría que hacer un análisis del mismo en tiempo de ejecución.
- Qué funciones se llaman, cuantas veces, desde donde, cuanto tiempo se invierte ejecutándolas.
- Para eso existen herramientas llamadas *profilers*.
- Análisis estadístico del código en ejecución.

Profiler **`gprof`**

- Compilar con la opción **`-pg`** para usar **`gprof`**
- Ejecutar.
- Se genera un archivo **`gmon.out`** que contiene la información de tiempos y llamadas a funciones.
- **`gprof -l <nombre del ejecutable> gmon.out`**
- Despliega el análisis estadístico del programa.
- Se pueden acumular:
  - Correr una vez. Luego mv gmon.out gmon.sum
  - Correr más veces y gprof -s executable-file gmon.out gmon.sum
  - Al final gprof executable-file gmon.sum > output-file

Salida de **`gprof`**

Se divide en dos partes:

- Flat profile.
- Call graph.

Flat profile

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
57.16	0.24	0.24				convolution
14.29	0.30	0.06	5506432	0.00	0.00	catsearch
11.91	0.35	0.05	1	50.02	50.02	readJPGImage
11.91	0.40	0.05				filterRB
2.38	0.41	0.01	1	10.00	10.00	readPNGImage
2.38	0.42	0.01				readAndCut
0.00	0.42	0.00	4	0.00	0.00	checkFileForRead
0.00	0.42	0.00	2	0.00	0.00	isValidUTC
0.00	0.42	0.00	1	0.00	0.00	calDateTime
0.00	0.42	0.00	1	0.00	0.00	julianDate
0.00	0.42	0.00	1	0.00	0.00	timeDayFrac

Call graph

```
index % time    self  children    called     name
                                                 <spontaneous>
[1]     57.1    0.24    0.00                 convolution [1]
-----------------------------------------------
                                                 <spontaneous>
[2]     16.7    0.01    0.06                 readAndCut [2]
                0.05    0.00       1/1           readJPGImage [5]
                0.01    0.00       1/1           readPNGImage [7]
-----------------------------------------------
                0.06    0.00 5506432/5506432     cloudcoverindex [4]
[3]     14.3    0.06    0.00 5506432          catsearch [3]
-----------------------------------------------
                                                 <spontaneous>
[4]     14.3    0.00    0.06                 cloudcoverindex [4]
                0.06    0.00 5506432/5506432     catsearch [3]
```

Profiler más detallado gcov

- Compilar con las banderas

  `-lgcov -fprofile-arcs -ftest-coverage`

- Ejecutar
- Usar **`gcov <nombre del archivo fuente>`**
- Luego **`less <nombre del archivo fuente>.gcov`**
- Despliega el número de veces que fue ejecutada cada línea del código.
- Se puede usar `ggcov` y `lcov` para ver la salida.

Salida de gcov

```
 5506432:465:int catsearch(int t, const int carr[],
                           int low, int upp) {
       -:  466:  int l, u, m;
 5506432:  467:  if ((low < upp) && (carr != NULL)) {
       -:  468:    l = low;
       -:  469:    u = upp;
34107891:  470:    while (l <= u) {
28601951:  471:      m = (int)((l + u) / 2);
28601951:  472:      if (carr[m] < t) l = m + 1;
16290435:  473:      else if (carr[m] > t) u = m - 1;
       -:  474:      else return m;
       -:  475:    }
 5505940:  476:    if (l <= upp)
 5497995:  477:      return l;
       -:  478:  }
       -:  479:  return -1;
       -:  480:}
```

```
 1:  547:unsigned int **convolution(int sdsz, int mv,
-:  548:                  unsigned int **img, int w, int h ) {
1:  549: int i, j, pelcolor, nesi = (int) ((double)sdsz / 2.0);
-:  550: int  n, m, nv;
1:  551: unsigned int **res = (unsigned int **) malloc(h *
-:  552:                              sizeof(unsigned int *));
1:  553: unsigned int *whole = (unsigned int *) malloc(h * w *
-:  554:                               sizeof(unsigned int));
```

```
    2649:  555:   for (i = 0; i < h; i++)
    2648:  556:      res[i] = whole + i * w;
       -:  557:
    2644:  558:   for (i = nesi; i < h - nesi; i++) {
 6990736:  559:      for (j = nesi; j < w - nesi; j++) {
 6990736:  560:         pelc = img[i][j];
 6990736:  561:         if (pelc == 0X00000000) {
 1484304:  562:            res[i][j] = 0X00000000;
       -:  563:         }
       -:  564:         else {
       -:  565:            nv = 0;
 33038592:  566:            for (n = i - nesi; n <= i + nesi; n++) {
165192960:  567:               for (m = j - nesi; m <= j + nesi; m++) {
137660800:  568:                  if ((img[n][m] ^ pelc) != 0) {
 2274157:  569:                     nv++;
       -:  570:                  }
       -:  571:               }
       -:  572:            }
 5506432:  573:            if (nv >= mv) res[i][j] = 0XFF000000|~pelc;
 5458616:  574:            else res[i][j] = 0XFF000000 | pelc;
       -:  575:         }
       -:  576:      }
       -:  577:   }
       1:  578:   return res;
       -:  579:}
```