Gcov on an embedded system

H Blasum, J Urban, F Görgen, SYSGO AG

16 Sept 2007

Terminology

- AFDX = avionics full-duplex ethernet
- PikeOS = microkernel for virtualization

Approach

- aim of the project: coverage for small embedded system (done), some code was reentrant, but parts also were inspectable manually (most code was non-reentrant initialization etc)
- aim of this talk: let's describe gcc/gcov and see how far we got/get, which questions arise, might be useful for next projects (and for others doing the same)
- philosophy: stripping down (not building up)

Structure

- GCC and gcov
- EMBedded
- PARallel

GCC: Coverage

- Extra code is made to test whether each chunk of code (more precise later) was run through
- Reasonable automatically testable security requirement in some safety norms, eg DO-178 (1992, avionics)
- avoid dead code (desired side effect: life code is also tested well)

$\longrightarrow$ coverage may be considered more "sw engineering" rather than pure "verification".

GCC: Gcov output: serial.c.gcov example

```
function serial_initSerial called 1 returned 100%
blocks executed 67%
        1: 126:{
        1: 127: unsigned int br;
        -: 128:
        1: 129:  br = onchip.sysclock / (16 * baudrate);
        -: 130:
        -: 131: /* limit BR */
        1: 132: if (br > 8191)
    #####: 133:   br = 8191;
        -: 134:
        1: 135: if (channel == 0) {
        -: 136: /* module A */
        1: 137: onchip.esci = &ESCI_A;
        -: 138:
```

GCC: Color by lcov

lcov -g gcov-3.4 –directory . –capture –output-file app.info
genhtml app.info

```
123                 : void serial_initSerial(
124                 :         int channel,
125                 :         int baudrate)
126               1 : {
127               1 :         unsigned int br;
128                 :
129               1 :         br = onchip.sysclock / (16 * baudrate);
130                 :
131                 :         /* limit BR */
132               1 :         if (br > 8191)
133               0 :                 br = 8191;
134                 :
135               1 :         if (channel == 0) {
136                 :                 /* module A */
137               1 :                 onchip.esci = &ESCI_A;
138                 :
139                 :                 /* enable pins */
147                 :         } else {
148                 :                 /* module B */
149               0 :                 onchip.esci = &ESCI_B;
150                 :
```

GCC: Lcov memory overview

LTP GCOV extension - code coverage report

Current view: directory - memgcov/collect
Test: app.info
Date: 2007-06-05
Code covered: 61.1 %
Instrumented lines: 1639
Executed lines: 1002

Filename	Coverage		
file1.c		31.1 %	19 / 61 lines
file2.c		100.0 %	5 / 5 lines
file3.c		100.0 %	19 / 19 lines
file4.c		0.0 %	0 / 18 lines
file5.c		96.2 %	50 / 52 lines
file6.c		71.9 %	159 / 221 lines
file7.c		75.0 %	84 / 112 lines
file8.c		4.0 %	4 / 99 lines
file9.c		81.2 %	56 / 69 lines
file10.c		75.0 %	45 / 60 lines

- lcov also can merge different runs
- also usable as an alternative to tcpdump/wireshark.

GCC: Coverage in gcc

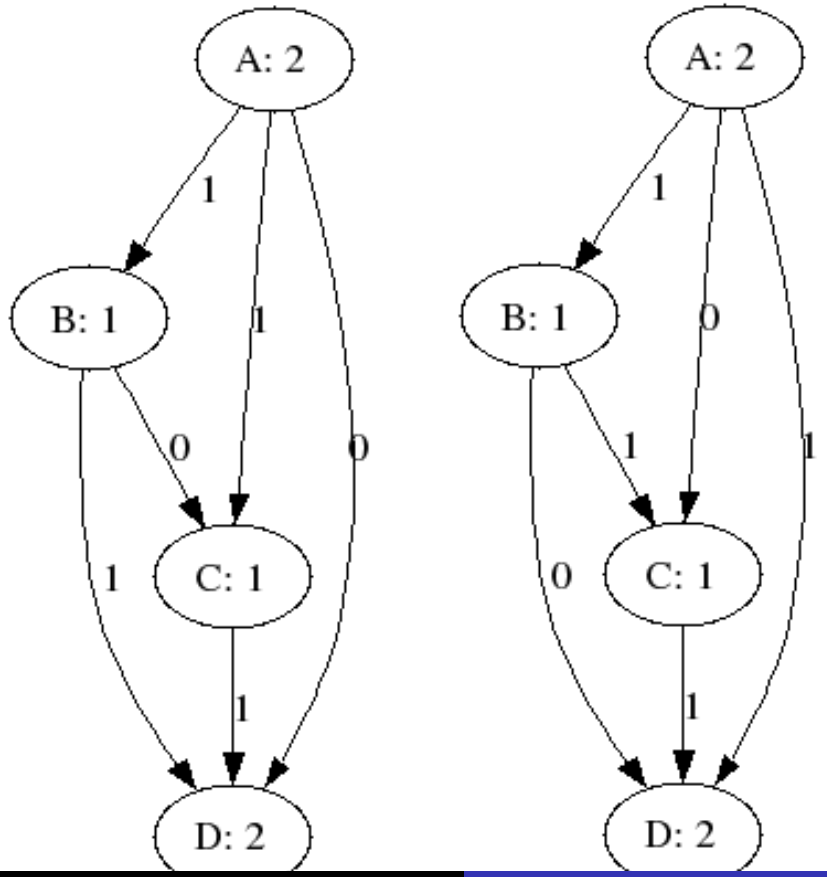
- Compilation phase: *gcc -O0 -o hello -fprofile-arcs -ftest-coverage hello.c (profile.c, coverage.c)*
- Data collection and extraction phase: *./hello*
- Reporting phase: *gcov hello.c*

(You have seen this pattern in the previous talk ...)

GCC: Terminology arcs and basic blocks

“ 而从 BB 的执行次数推算出每个arc 的执行次数是困难的 ”

A block is a set of instructions executed together (no jumps, calls inside in assemby).
Observation on connected directed graphs: edges (arcs) preserve more (or the same) information than nodes (basic blocks).



GCC: hello.gcno

```
00000000  6f 6e 63 67 2a 34 30 33  9f ef 9d b9  00 00 00 01  |oncg*403.ï.¹....|
00000010   09 00 00 00 03 00 00 00  d7 8b 29 2b 02 00 00 00  |........×.)+....|
00000020   6d 61 69 6e 00 00 00 00  02 00 00 00 68 65 6c 6c  |main........hell|
00000030   6f 2e 63 00 03 00 00 00  00 00 41 01 07 00 00 00  |o.c.......A.....|
00000040   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050   00 00 00 00 00 00 00 00  00 00 00 00  00 00 43 01  |..............C.|
00000060   03 00 00 00 00 00 00 00  01 00 00 00 05 00 00 00  |................|
00000070   00 00 43 01 03 00 00 00  01 00 00 00 02 00 00 00  |..C.............|
00000080   05 00 00 00 00 00 43 01  05 00 00 00 02 00 00 00  |......C.........|
00000090   03 00 00 00 04 00 00 00  05 00 00 00 00 00 00 00  |................|
000000a0   00 00 43 01 05 00 00 00  03 00 00 00 06 00 00 00  |..C.............|
000000b0   03 00 00 00 04 00 00 00  05 00 00 00 00 00 43 01  |..............C.|
000000c0   03 00 00 00 04 00 00 00  02 00 00 00 00 00 00 00  |................|
000000d0   00 00 43 01 03 00 00 00  05 00 00 00 06 00 00 00  |..C.............|
000000e0   05 00 00 00  00 00 45 01  0a 00 00 00 01 00 00 00  |......E.........|
000000f0   00 00 00 00 02 00 00 00  68 65 6c 6c 6f 2e 63 00  |........hello.c.|
00000100   03 00 00 00 04 00 00 00  05 00 00 00 00 00 00 00  |................|
00000110   00 00 00 00 00 00 45 01  04 00 00 00 03 00 00 00  |......E.........|
00000120   06 00 00 00 00 00 00 00  00 00 00 00 00 00 45 01  |..............E.|
00000130   04 00 00 00 04 00 00 00  05 00 00 00 00 00 00 00  |................|
00000140   00 00 00 00 00 00 45 01  04 00 00 00 05 00 00 00  |......E.........|
00000150   08 00 00 00 00 00 00 00  00 00 00 00              |............|

0000015c
```

GCC: hello.gcda

```
00000000 61 64 63 67 2a 34 30 33 9f ef 9d b9  00 00 00 01|adcg*403.ï.¹....|
00000010  02 00 00 00 03 00 00 00 d7 8b 29 2b  00 00 a1 01|........×.)+..¡.|
00000020  06 00 00 00 0a 00 00 00 00 00 00 00 01 00 00 00|................|
00000030  00 00 00 00 0a 00 00 00 00 00 00 00  00 00 00 a1|............¡|
00000040 09 00 00 00 00 00 00 00 03 00 00 00 01 00 00 00 |................|
00000050 15 00 00 00 00 00 00 00 0a 00 00 00 00 00 00 00 |................|
00000060 0a 00 00 00 00 00 00 00 00 00 00 a3 09 00 00 00 |...........£....|
00000070 e8 ed 3a cf 03 00 00 00 01 00 00 00 15 00 00 00 |èí:Ï............|
00000080 00 00 00 00 0a 00 00 00 00 00 00 00 0a 00 00 00 |................|
00000090 00 00 00 00 |....|

00000094
```

EMB: Embedded special points

- no dynamic memory allocation
- no file system
- $\longrightarrow$ no libc (fseek, fopen, fclose, fread, fwrite, malloc, realloc)
- $\longrightarrow$ requires some attention to detail, but straightforward in principle (see paper)

EMB: Surprise: gcov uses C++ constructors

Action	i386 (gcc-3.3)	i386 (gcc-3.4)	PPC target
Allocate memory for counters	.ctors→ _GLOBAL__I _main_GCOV →__bb_init_func (→atexit)	.ctors→ _GLOBAL__I _main_GCOV →__gcov_init (→atexit)	p4_gcov_entry →.ctors→ _GLOBAL__I _main_GCOV →__gcov_init
(C) pointer to storage area as argument to init function	struct bb *blocks (0x804ade8)	struct gcov_info *p (0x804b5c0)	hardcoded
Reading phase: harvest data	atexit→ __bb_exit_func	atexit→ __gcov_exit	set IP to __gcov_exit

EMB: Entry point p4_gcov_entry

From *init_board (cboot.c)* run:

```
extern typedef void (*Xtor) (void);
extern Xtor const __CTOR_LIST__[];
extern Xtor const __CTOR_END__[];
void p4_gcov_entry(){
Xtor const *ptr = __CTOR_LIST__;
Xtor const *end = __CTOR_END__;
for( ; ptr != end ; ptr++) {
   if(*ptr) {
      (*ptr)();
   }
}
}
```

EMB: Purging libgcov glibc dependencies: stdlib

```
void *mmalloc (size_t size) {
    long mem_back = mem_ctr;
    mem_ctr += size;
    return ((void*)((unsigned)mem_ptr + (unsigned)mem_back));
}
```

where

```
mem_ptr = (void*)0x09000000;
```

Other (obvious) memory replacements: mmemcpy, mmemset, mmemcmp. A realloc in the code can be avoided.

EMB: Purging libgcov glibc dependencies: stdio

Define QDOS-style file system:

```
struct gcdaone {
  char *filename;
  int mem_begin; } __attribute__ ((__packed__));
struct gcdaall {
  int checksum[4];
  int memsize;
  int filecount;
  struct gcdaone **file;
} __attribute__ ((__packed__));
```

EMB: Purging stdio ctd.

Assume files are operated on sequentially. Opening:

```
gcov_open() {
k = 0;
while (*name && k<FILENAMELEN-1) {
  gcdaall_ptr->file[file_ctr]->filename[k] = *name; k++; name++; }
gcdaall_ptr->file[file_ctr]->filename[k] = 0;
gcdaall_ptr->file[file_ctr]->mem_begin = mem_ctr;
}
```

Writing:

```
static void gcov_write_block (unsigned size) {
   mmemcpy((void*)((unsigned)mem_ptr + (unsigned)mem_ctr),
     gcov_var.buffer, size << 2);
   mem_ctr += size << 2;
}
```

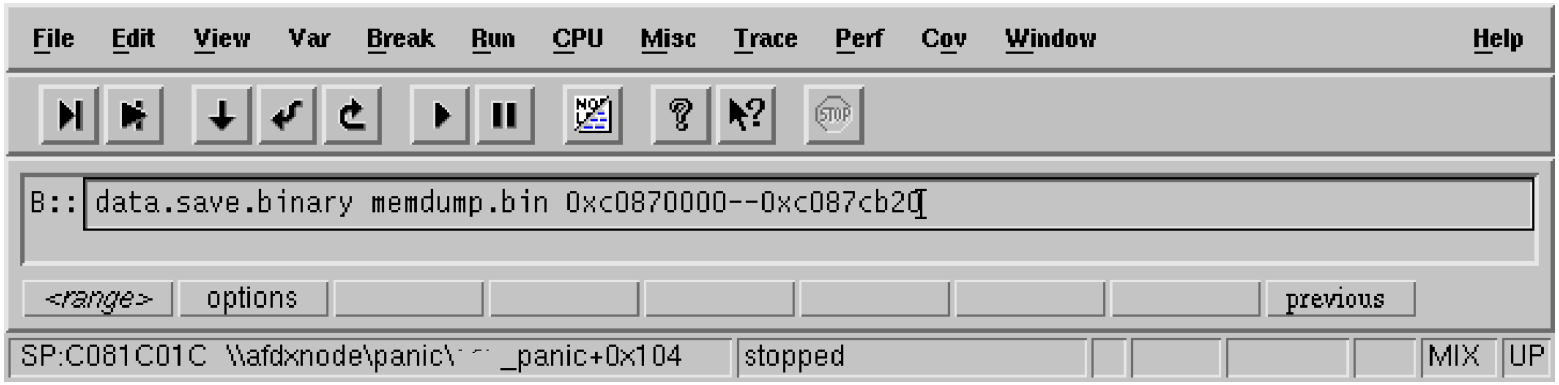
Closing simply increments *file_ctr*. *Fseek* can be omitted.

EMB: Handling gcno crc stamps

Ok, there is a little nastiness in mem2gcda (byte 9 to 12 of gc(no|da) files are a crc checksum):

- Usually at collection time libgcov reads crc stamps from the gcno files (e.g. to detect intermittent recompilation).
- Option: ignore this, and on the host mem2gcda simply simulate the crc stamps.
- Option: give list of gcno stamps as argument to libgcov compilation.

EMB: Post mortem: Dumping the memory



- tell system to stop when you want it
- set IP to gcov_exit (e.g. in Lauterbach debugger)
- see what is in memory

EMB: Laziness: feeding entire memdumps

Some short convenience scripts:

- *gcov_check*: check a gcov file for negative counts (these can result from *solve_flow_graph* when the graph solver is fed garbage)
- *gcov_dump:* generate bunch of gcov files from dump, uses *extract_mem2gcda, mem2gcda*, *gcov_rec* (recursively generate gcov files) and *gcov_check*

EMB: Review: Recipe chosen

- Instrument (printf) gcov library to understand *gcov* in gcc sources.
- Get rid of file system dependence on I386.
- Get rid of libc dependence on I386.
- Port to PPC.
- Play with memory output (gcov/lcov/gcov_dump).

PAR: Is it reentrant?

```
28: 3d 20 00 00 lis r9,0
2c: 39 69 00 10 addi r11,r9,16
30: 81 2b 00 00 lwz r9,0(r11)
34: 81 4b 00 04 lwz r10,4(r11)
38: 31 4a 00 01 addic r10,r10,1
3c: 7d 29 01 94 addze r9,r9
40: 91 2b 00 00 stw r9,0(r11)
44: 91 4b 00 04 stw r10,4(r11)
48: 3d 20 00 00 lis r9,0
4c: 38 69 00 00 addi r3,r9,0
50: 48 00 00 01 bl 50 <main+0x50>
54: 81 3f 00 08 lwz r9,8(r31)
58: 38 09 00 01 addi r0,r9,1
5c: 90 1f 00 08 stw r0,8(r31)
60: 3d 20 00 00 lis r9,0
64: 39 69 00 10 addi r11,r9,16
68: 81 2b 00 00 lwz r9,0(r11)
6c: 81 4b 00 04 lwz r10,4(r11)
70: 31 4a 00 01 addic r10,r10,1
74: 7d 29 01 94 addze r9,r9
78: 91 2b 00 00 stw r9,0(r11)
```

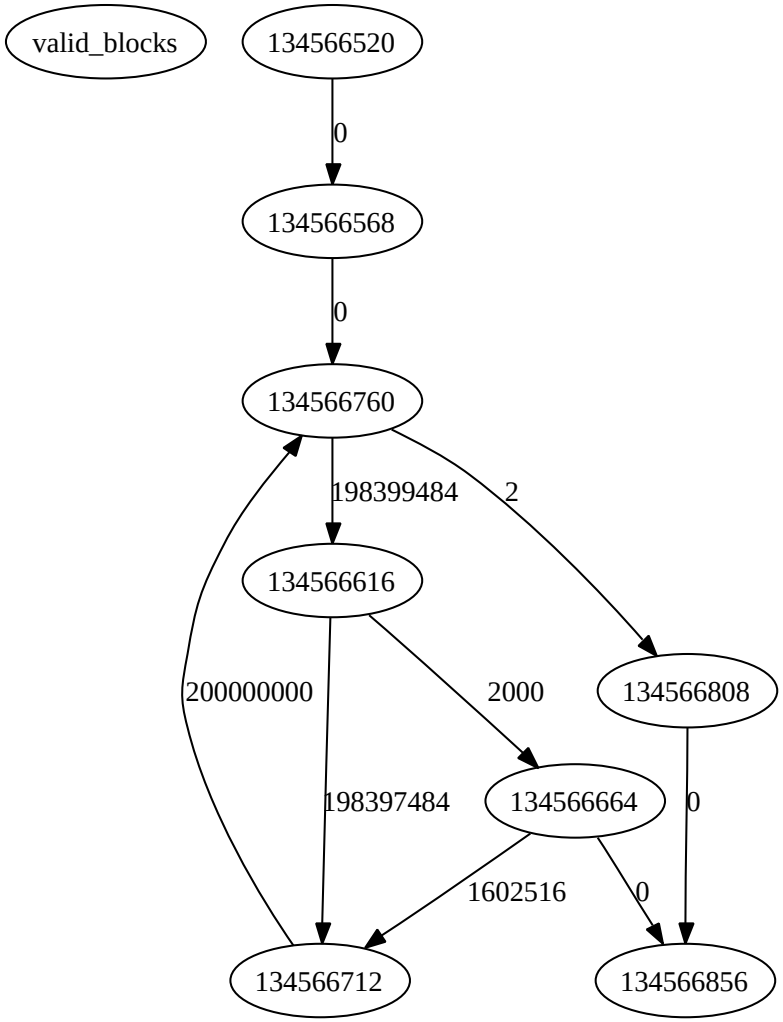
PAR: Pthread.c main function

```
-: 17:int main ()
1: 18:{
-: 19: /* create pthreads and wait for them */
-: 20: pthread_t t1, t2;
1: 21: pthread_create (&t1, NULL, worker, NULL);
1: 22: pthread_create (&t2, NULL, worker, NULL);
1: 23: pthread_join (t1, NULL);
1: 24: pthread_join (t2, NULL);
1: 25: return 0;
-: 26:}
```

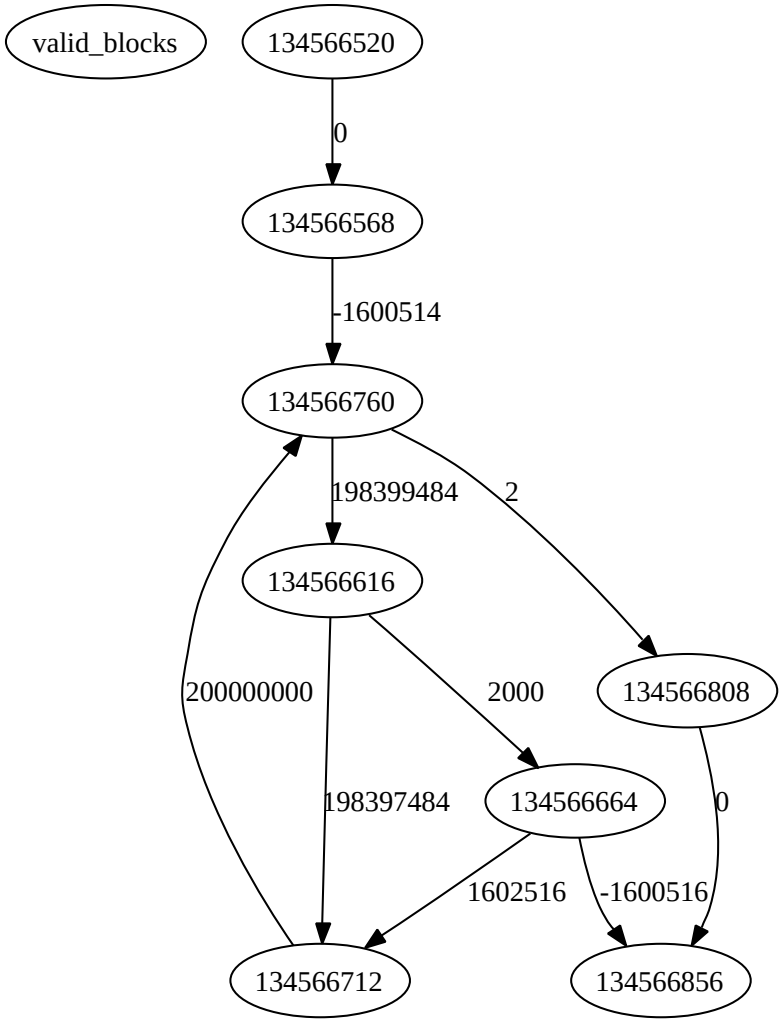
PAR: Pthread.gcov.c worker

```
        -:  1:#include <pthread.h>
        -:  2:#include <stdio.h>
        -:  3:#include <sys/io.h>
        -:  4:
 -1600514:  5:void *worker (void *args) {
        -:  6:  int i;
 -1600514:  7:  int j = 0;
198399486:  8:  for (i = 0; i< 100000*1000; i++) {
198399484:  9:    j++;
198399484: 10:     if (j % 100000 == 0) {
     2000: 11:       printf("j now is %d.\n", j);
        -: 12:     }
        -: 13:   }
        -: 14:
        2: 15:  }
        -: 16:
```

PAR: Before subtraction



PAR: After subtraction



PAR: Networking flow

- you can subtract a spanning tree from a flow graph (e.g. Knuth, AoC)
- on a good flow graph this works fine
- when irregularities occur, this fails

PAR: What one can do...

- disable flow graph (tried and tested)
- buffer for each thread (Bartolini/Prete gprof way)

PAR: Simple and elegant

- subtract the "correct" spanning tree (prerequisite: make sure that structure of bb ordering is maintained)

```
/* Follow successors of blocks, and register these edges.  */
FOR_BB_BETWEEN (bb, ENTRY_BLOCK_PTR, EXIT_BLOCK_PTR, next_bb)
  FOR_EACH_EDGE (e, ei, bb->succs)        i
    elist->index_to_edge[num_edges++] = e;
```

PAR: Current entry point

(one is coming from)

```
 bt #0  branch_prob () at ../../gcc/gcc/gcc/profile.c:759 #1
0x083a03e1 in tree_profiling ()
at ../../gcc/gcc/gcc/tree-profile.c:420 #2  0x082a0f8f in
execute_one_pass (pass=0x8826120)
at ../../gcc/gcc/gcc/passes.c:1108 #3  0x082a118f
in execute_pass_list (pass=0x8826120)
at ../../gcc/gcc/gcc/passes.c:1161 #4
0x084fee73 in cgraph_analyze_function (node=0xb7b0e600)
at ../../gcc/gcc/gcc/cgraphunit.c:773 #5  0x084ffd5d in
cgraph_finalize_function (decl=0xb7b0e580, nested=0 '\0')
at ../../gcc/gcc/gcc/cgraphunit.c:542
```

Thank you for your attention.

- EMB: would like to see this on more systems (some willingness to support)
- PAR: to check whether order is always preserved before profile