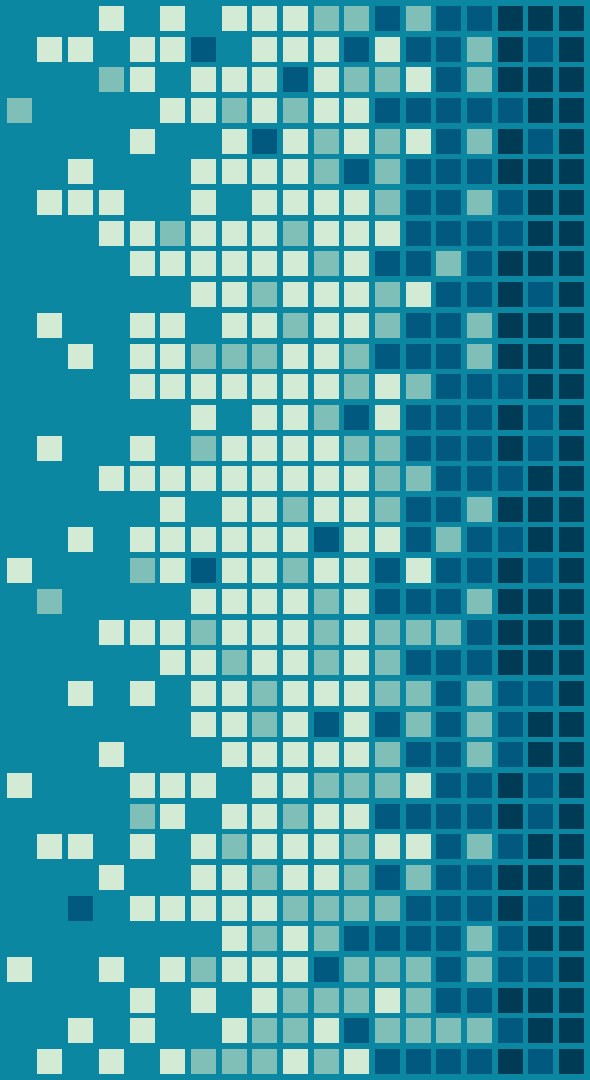
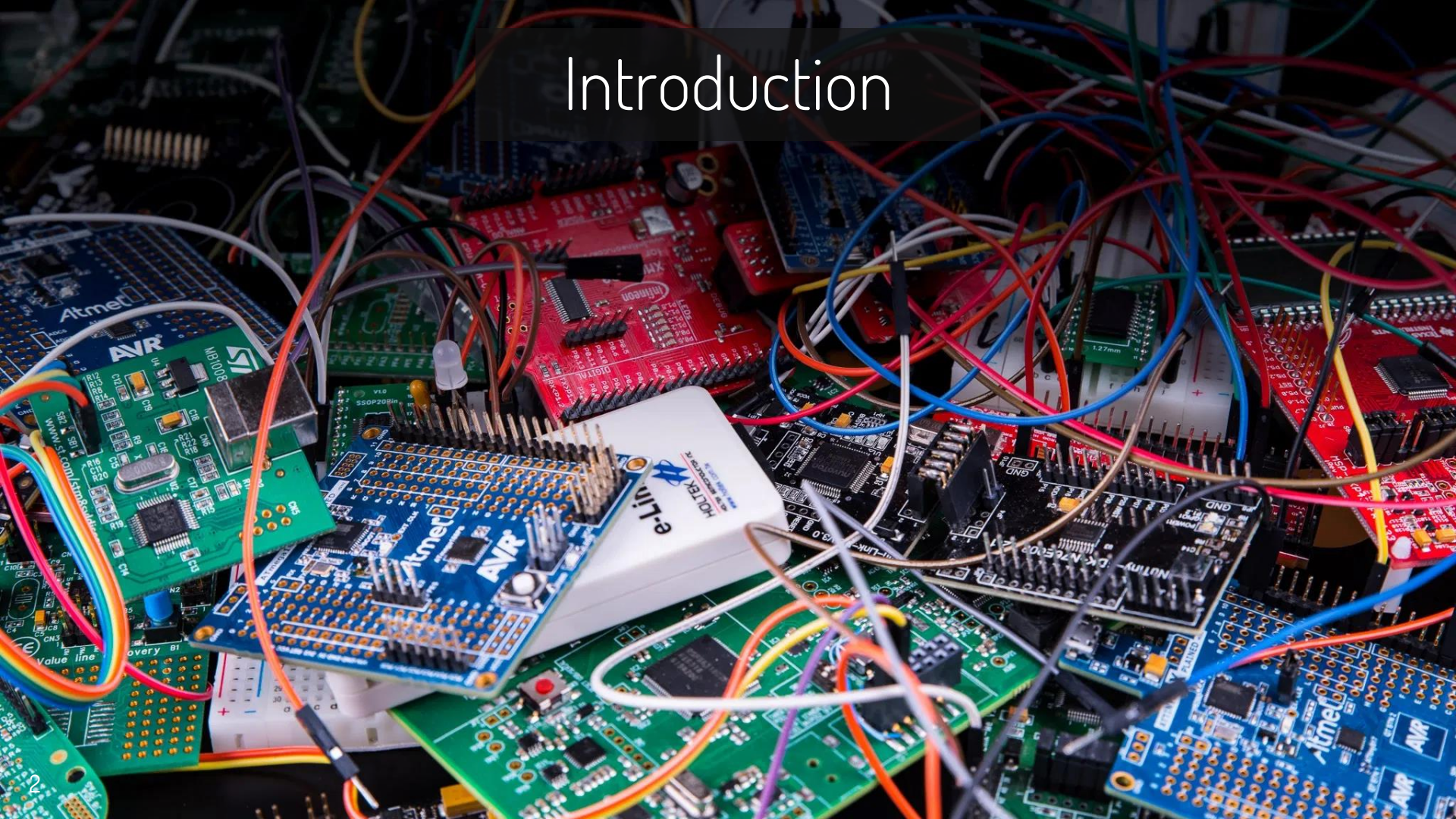
A Low Footprint gprof-based Profiler for Microcontrollers

Master's Thesis Defense

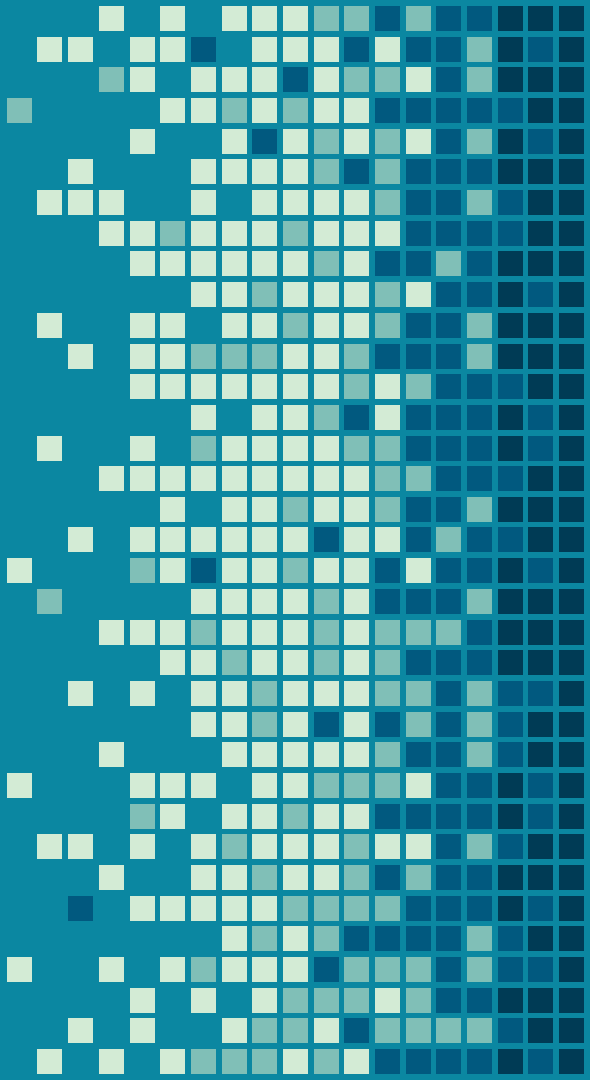
Michael D'Argenio – mjdargen@ncsu.edu
May 4, 2020 1:00pm



Introduction



PROFILING

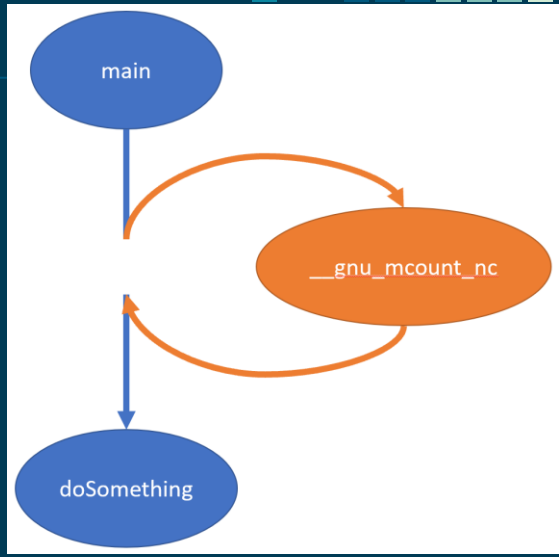


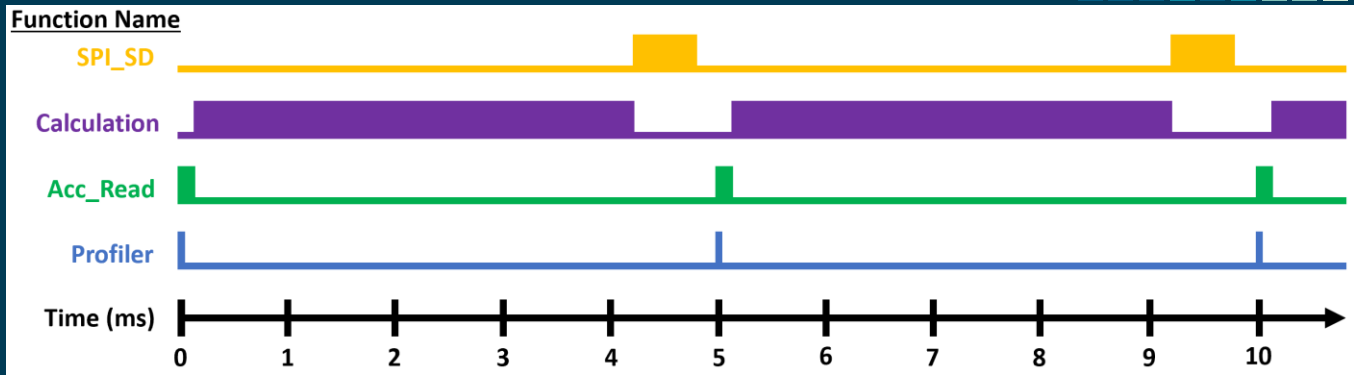
Profiling

- A form of dynamic program analysis to record and analyze metrics during a program's execution.
  - % execution time
  - # of executions
  - Memory usage
  - Energy consumption
  - More
- Profiling used to identify optimization opportunities.
  - Identify hot spots in executions.
  - Leave trail to understand how program executed.

Profilers

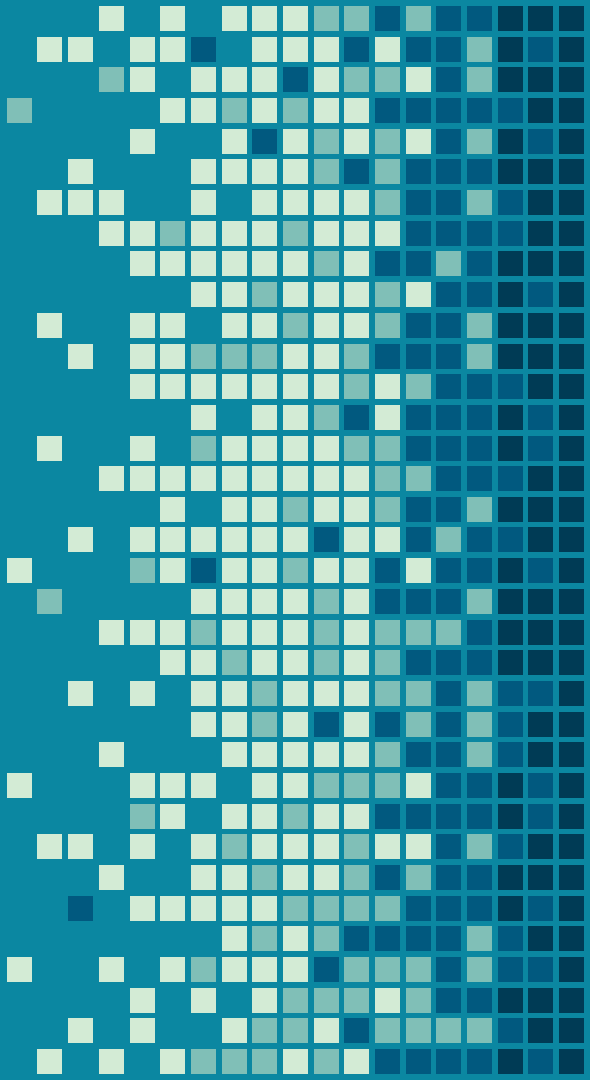
- Output Types
  - Flat profile
  - Call graph profile
  - Annotated instructions
- Methodologies
  - Instrumentation
  - Statistical
  - Event-based
  - Simulation





"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he [they] will be wise to look carefully at the critical code; but only after that code has been identified. It is often a mistake to make a priori judgments about what parts of a program are really critical, since the universal experience of programmers who have been using measurement tools has been that their intuitive guesses fail."

-Donald Knuth

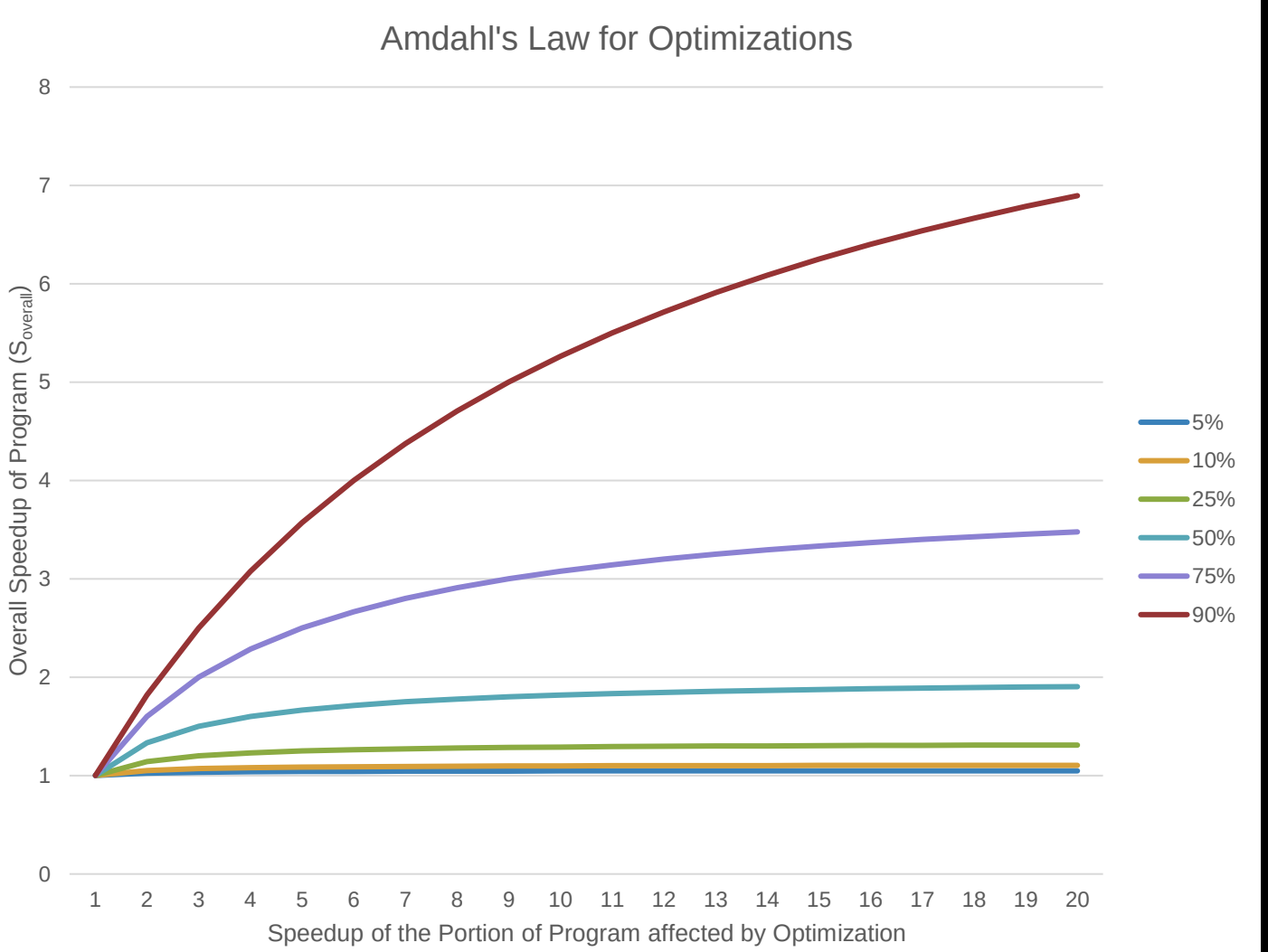


Amdahl's Law Applied to Optimizations

- $S_{overall}$ - theoretical overall speedup of the optimized program.
- s - speedup of the particular optimized portion of the program.
- f - fraction of execution time benefitting from the optimization.

$$S_{overall}(s) = \frac{1}{(1-f) + \frac{f}{s}}; \qquad \lim_{s \to \infty} S_{overall}(s) = \frac{1}{(1-f)}$$

- Optimization A speeds up function Z by a factor of 5.
- Function Z consumes 25% of the program's execution time.
- ∴ , Optimization A provides an overall speedup of 1.25.


Amdahl's Law for Optimizations
Overall Speedup of Program ($S_{overall}$)
0
1
2
3
4
5
6
7
8
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Speedup of the Portion of Program affected by Optimization
5%
10%
25%
50%
75%
90%

Paradox of Profiling Microcontrollers

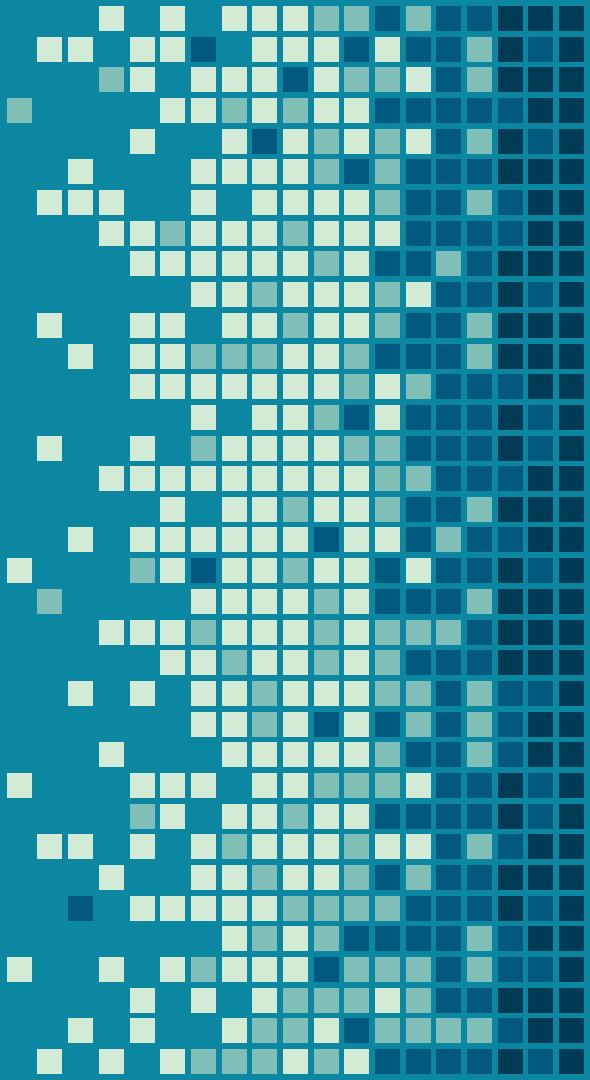
- MCUs are intended for a singular, special purpose.
- ∴ , MCUs have limitations: processor speed, pipeline capabilities, instruction sets, functional units, memory capacity, power, etc.
- These limitations keep the cost low and make MCUs useful.

Paradox:

Profiling allows us to make better use of limited resources; however, these limited resources make profiling impossible.

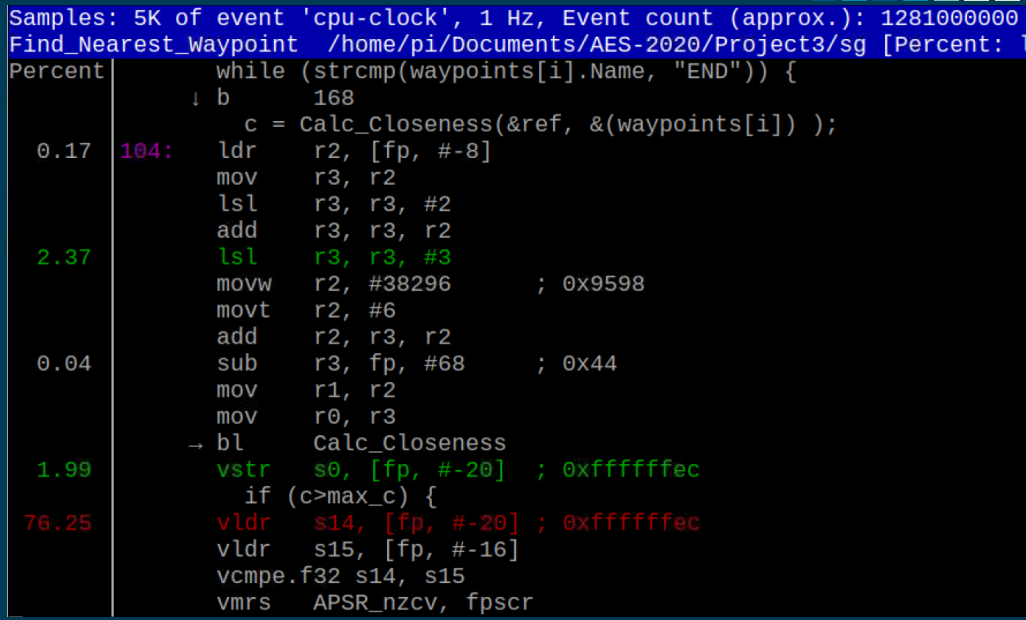
- Existing profilers require too much memory and processing.
- Need to investigate new solutions to optimize MCUs.

RELATED WORK



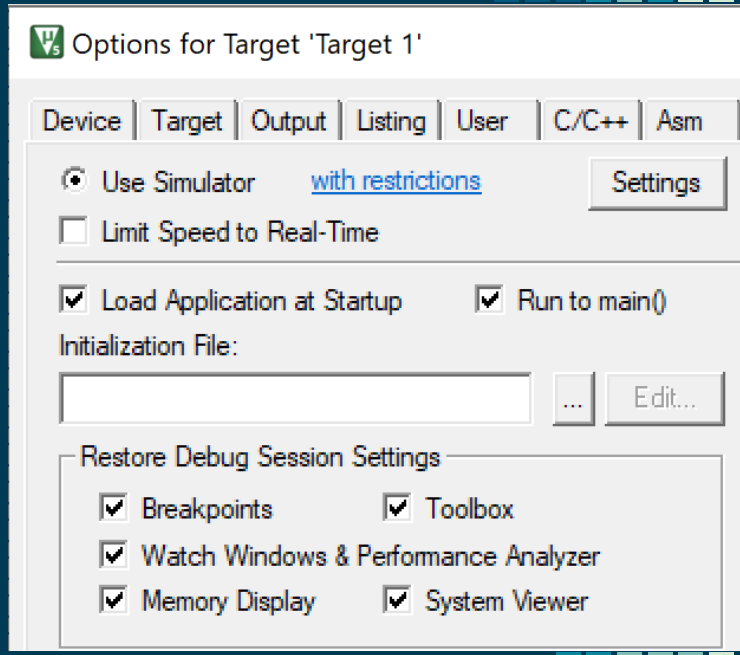
Existing Profilers

- Existing profilers: Valgrind, gprof, perf, OProfile, and more.
- Work for embedded Linux platforms.
- Not suitable for microcontrollers as is.
- Requires OS to provide supervision of executable.
- Needs full standard C library.

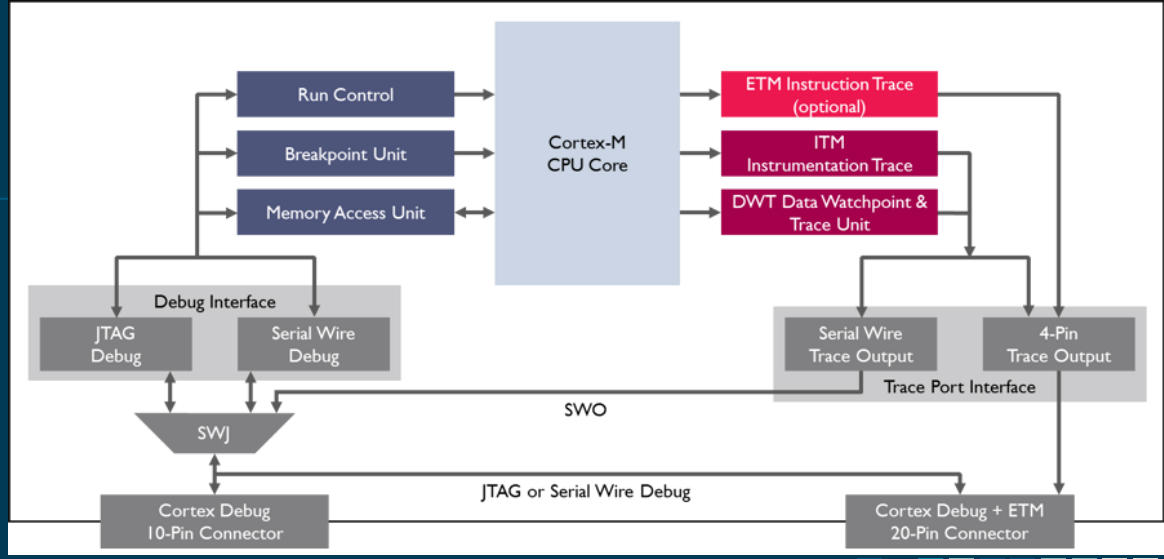


Simulators

- Provides significant insight into how program executes.
- This level of abstraction can introduce significant error.
- No interaction or connection with real-world peripherals or external hardware.
- Not necessarily representative.
- Helpful for debug, not for profile-guided optimization.



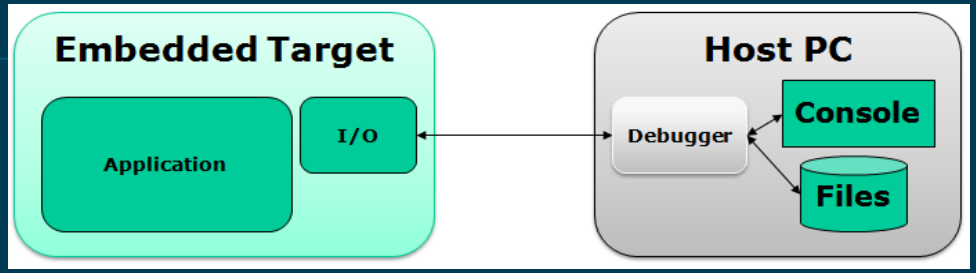
Tracing

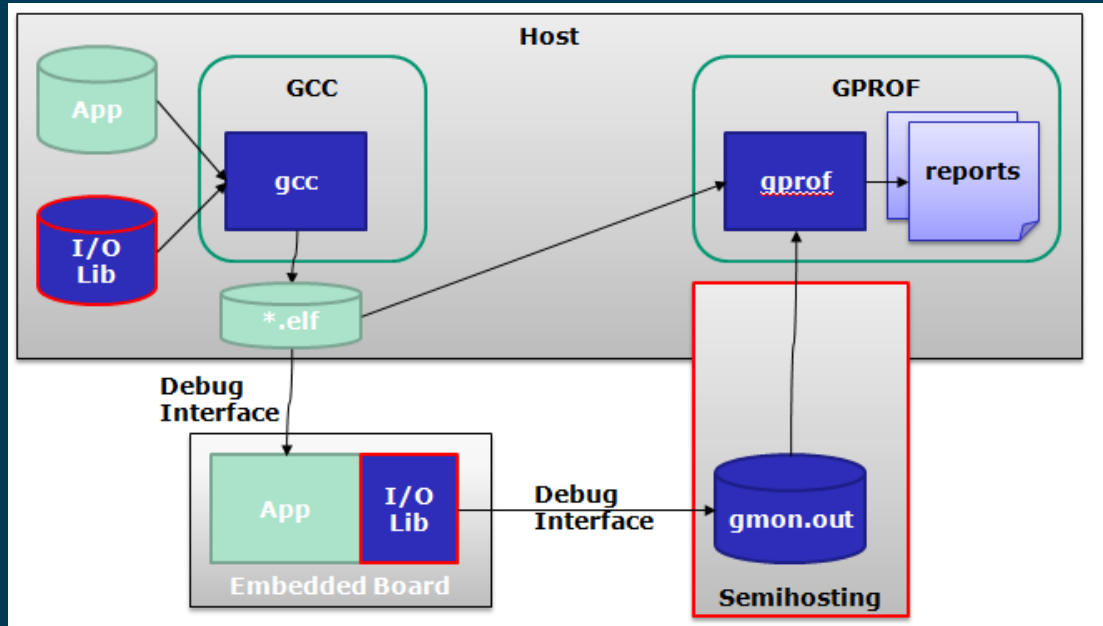


- Provides near infinite amount of detail.
- Records every instruction executed.
- Functionality varies widely across implementations.
- May require extra manual work to extract pertinent information.
- Tracing often reserved for higher-end microcontrollers.
- Can require costly external debug technology.

Semihosting gprof

- Specific to ARM Cortex-M.
- Requires debug semihosting interface for I/O file transfer.
- Uses a significant amount of memory
- Exp. 4 kB Program consumes 9 kB of RAM.





MCU Profiler Comparison

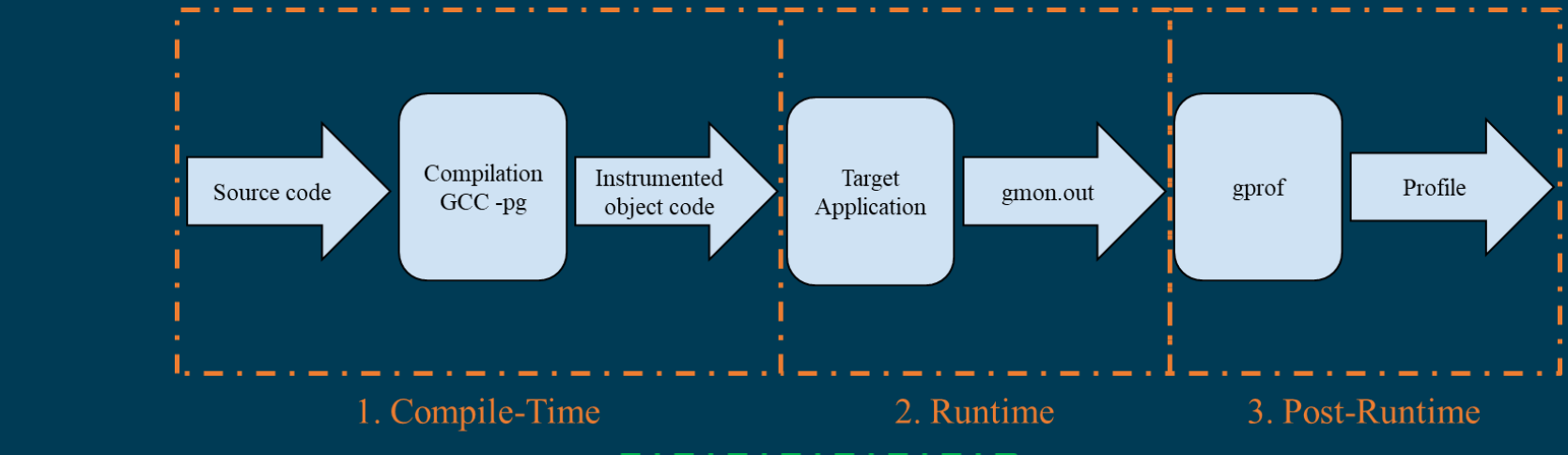
	ROM Usage	RAM Usage	CPU Performance	Expressive Profile	Indicative of Runtime Execution	Ease of Use	Cross-Platform Support
Existing Profilers	✗	✗	✗	✓	✓	✓	✓
Simulators	N/A	N/A	N/A	✓	✗	✗	✗
Tracing	✗	✗	✗	✓	-	-	-
Semihosting gprof	✓	✗	✓	-	✓	-	-
Thesis - gprof	✓	✓	- *	-	✓	- *	✓

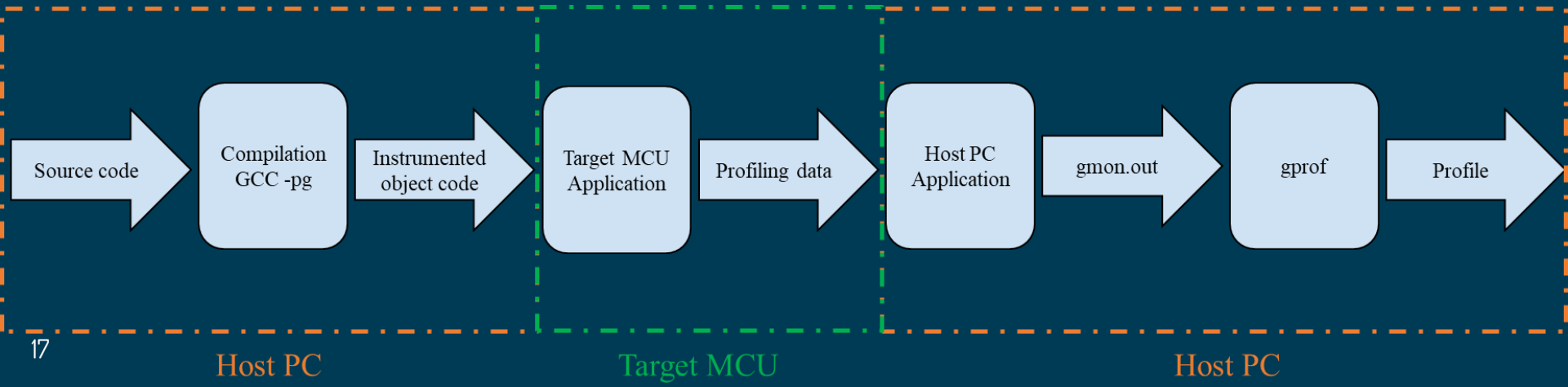
✓ Positive - Neutral ✗ Negative

*solution could be further improved in this area by future work

SYSTEM DESIGN

System Overview





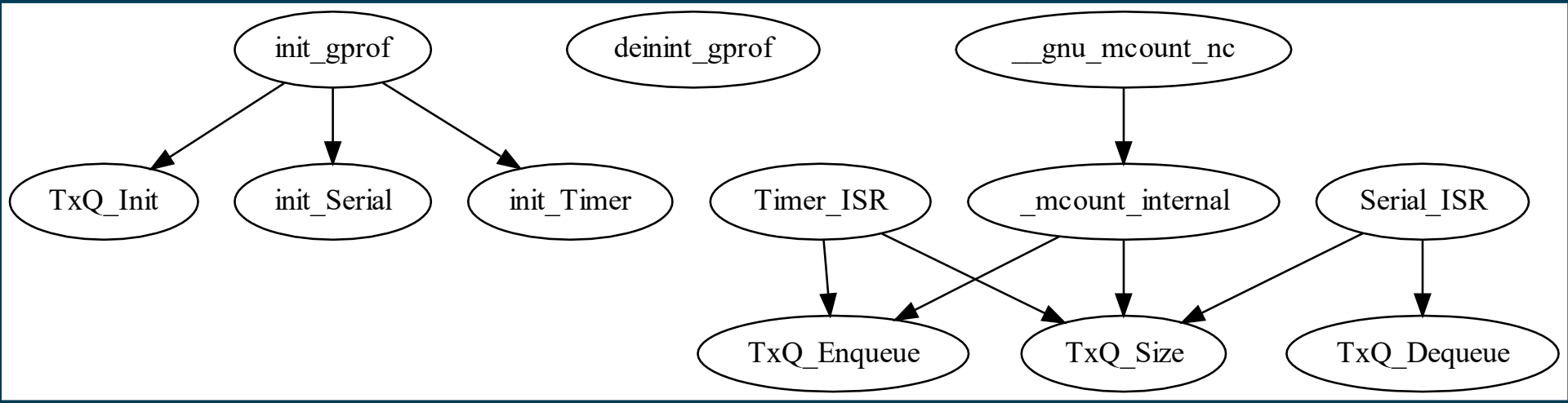
System Overview

- **Compile-time**: GCC adds instrumented code to the program to record profiling data.
- **Runtime**: Program executes. Profiling data is captured.
  - **MCU**: Program executes. Instrumented instructions transmit profiling data via serial interface to the host PC.
  - **PC**: Application receives profiling data via serial interface throughout target MCU's program execution. Constructs gmon.out file upon program exit.
- **Post-Runtime**: Input executable file and gmon.out to gprof. gprof generates profile.

Target MCU Application

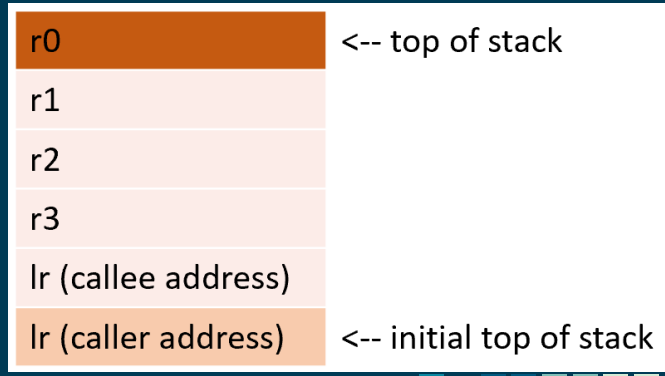
- Call Graph Arc Sampler – capture callee/caller addresses
  - mcount.S
  - gmon_arc.c
- Program Counter Sampler – periodically capture program counter
  - gmon_profil.c
- Transmit Queue – buffer structure for data transmission
  - gmon_queue.c
- Serial Interface – framework for transmitting data to PC
  - gmon_serial.c

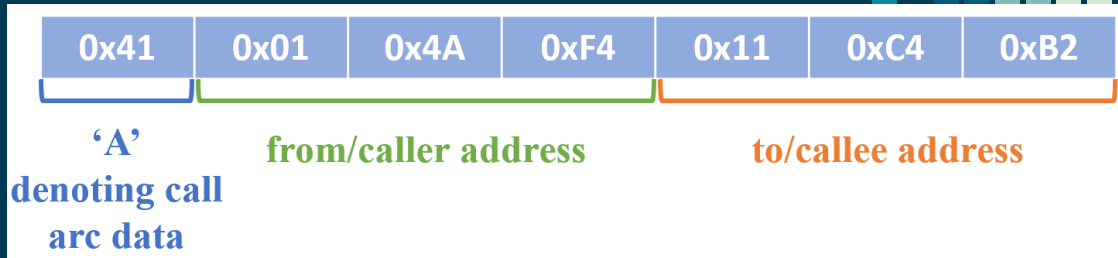
Target MCU Application



Call Graph Arc Sampler

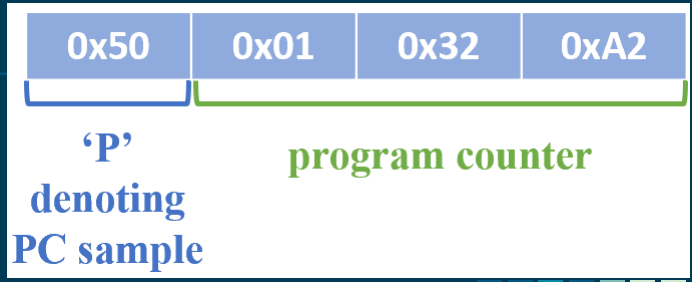
- __gnu_mcount_nc() - stub to retrieve callee/caller addresses
  - GCC inserts __gnu_mcount_nc() between every call
  - Implemented in assembly
  - Collects caller and callee address from stack
  - Passes to _mcount_internal()
- _mcount_internal() - call arc serial transmission
  - Passes addresses to transmit queue
  - Prevent data contamination
- init_gprof()
  - starts profiling
- deinit_gprof()
  - stops profiling





Program Counter Sampler

- init_Timer() – initializes periodic timer interrupt
  - Provides periodicity for sampling
  - MCU specific
  - Configurable sampling rate
  - Highest priority
- Timer_ISR() – timer interrupt service routine
  - Retrieves program counter from stack
  - Multiple stack pointers (SP)
  - Define SP at compile-time
  - Calculate frame size and offset




```
// retrieve stack pointer
// add additional frames on stack to retrieve pc
sp = (CUR_SP + FRAME_SIZE + HW_RET_ADX_OFFSET);
// retrieve program counter
pc = *(uint32_t*)(sp);
```

```
// Comment out USING_RTOS definition if not using RTOS
// #define USING_RTOS
#define HW_RET_ADX_OFFSET (24)
#define IRQ_FRAME_SIZE (8)
#define PC_OFFSET (24)

#ifdef USING_RTOS // Don't need these since PC is on PSP, not MSP
  #define FRAME_SIZE  (0)
  #define CUR_SP      (__get_PSP())
#else // Using MSP, so stack frames are also on the MSP stack
  #define FRAME_SIZE  (IRQ_FRAME_SIZE + PC_OFFSET)
  #define CUR_SP      (__get_MSP())
#endif
```

Transmit Queue

- Static queue structure for safe access.
- Queue used for serial port transmit interrupt.
- Queue involves critical sections of code.
  - Store state & disable interrupts before modifying queue.
  - Restore state after modifying queue.
- TxQ_Init() – initializes transmit queue.
- TxQ_Size() – determines current size of queue.
- TxQ_Enqueue() – enqueues a byte of data.
- TxQ_Dequeue() – dequeues a byte of data.

```
static uint8_t TxQ[MAX_Q_SIZE]; /* ring queue data buffer */
static uint16_t TxQHead; /* queue head index */
static uint16_t TxQTail; /* queue tail index */
static uint16_t TxQSize; /* queue size data */
```

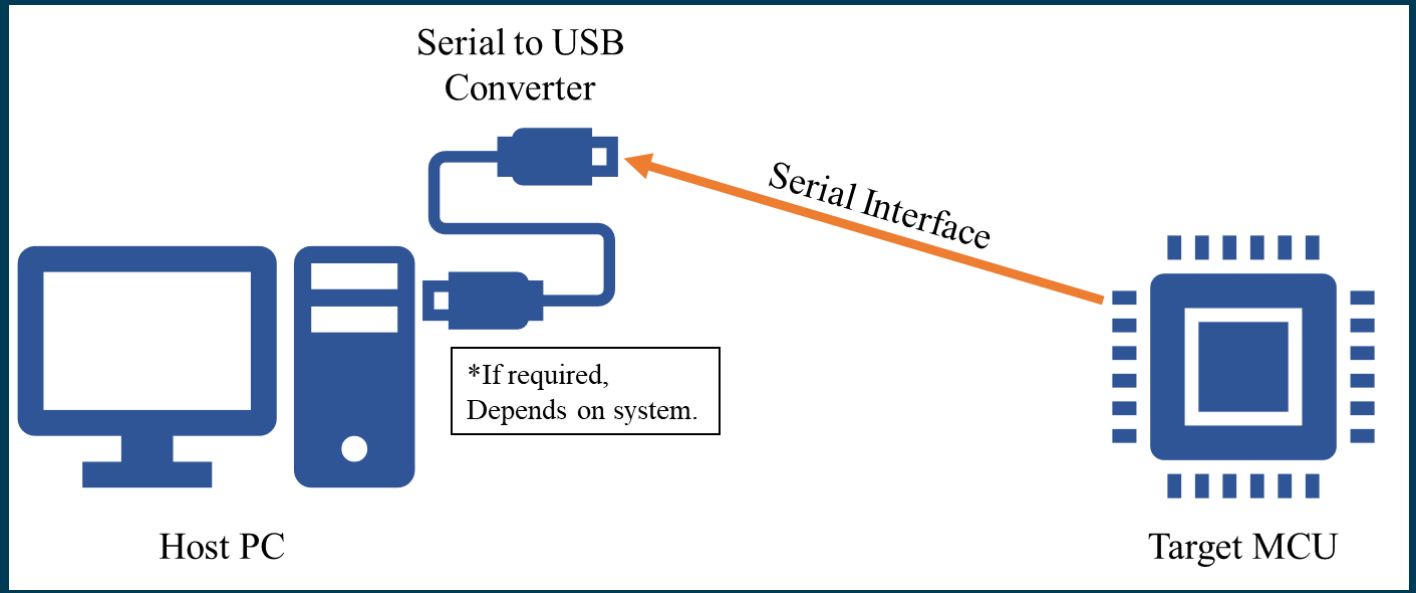
```
uint8_t TxQ_Enqueue(uint8_t data) {

  // if queue is full, return error
  if (TxQSize==MAX_Q_SIZE) {
    return 0; // failure
  }
  // else we can enqueue new element
  else {
    // save current masking state
    // disable interrupts
    EnterCritical();
    // store data, increment tail
    TxQ[TxQTail++] = data;
    TxQTail %= MAX_Q_SIZE;
    // increment size
    TxQSize++;
    // restore interrupt masking state
    ExitCritical();
    return 1; // success
  }
}
```

Serial Interface

- MCU dependent – consider following variables:
  - Available interfaces: UART, SPI, I2C, USB, or other.
  - Maximal throughput to keep up with profiler data.
  - Polling vs. Interrupt
- Exp. UART operating 1.5 MegaBaud with transmit interrupt
- init_Serial() – initializes serial port and interrupt.
- Serial_ISR() – transmit data register empty ISR.
  - Call TxQ_Dequeue() in interrupt to retrieve data.
  - Possibly disable ISR if queue is empty, re-enable later.

Serial Receiver

- USB to Serial Converter
- Serial Monitor/Capture Program

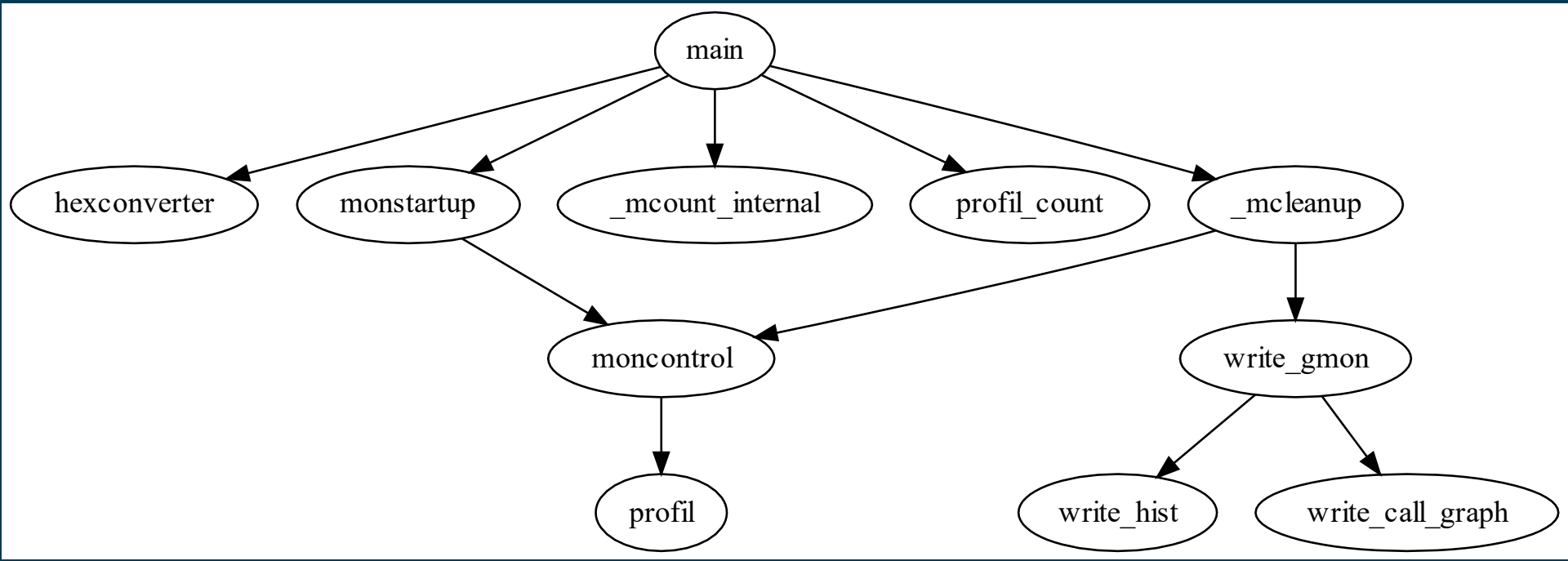


Host PC Application

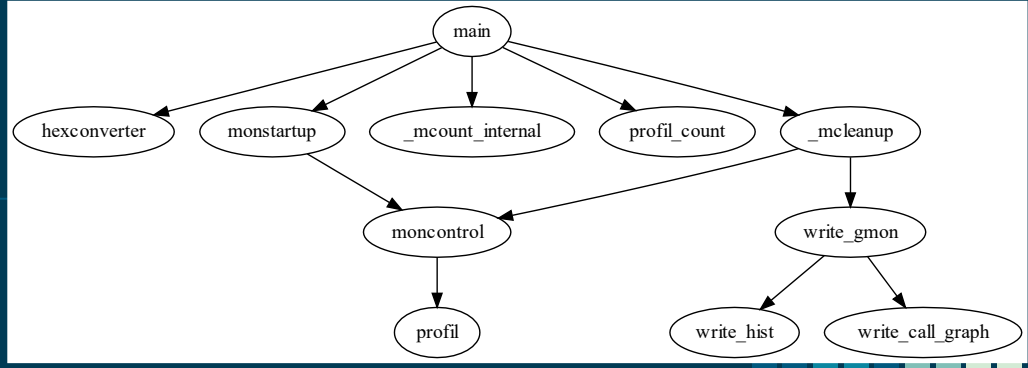
- Main/While – input data parsing and process workflow
  - main.c
- gmon Handling – file i/o processing for gmon.out
  - gmon.c
- Call Graph Arc Processing – call graph construction
  - mcount.c
- Program Counter Processing – histogram construction
  - profil.c

*Adapted from GNU C Library (glibc) version 2.30.

Host PC Application



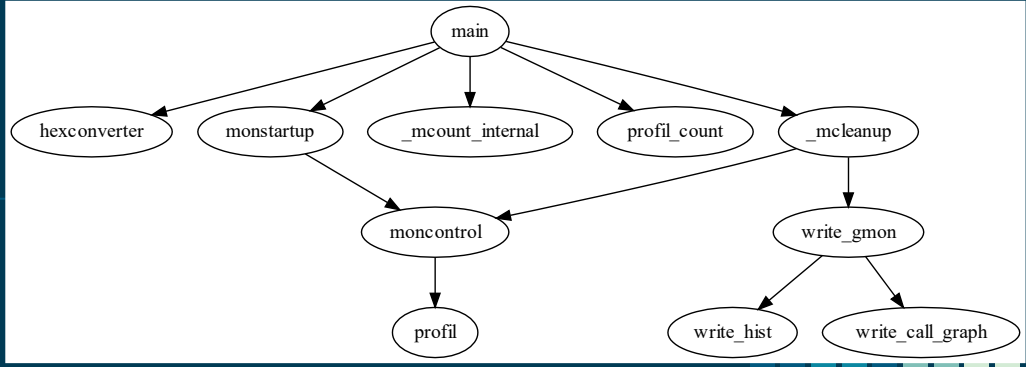
Main/While



- main() – main workflow
  - Calls monstartup() to initialize profiling
  - Opens input file with profiler samples
  - while() loop processes profiler samples until end of file
    - Calls profil_count() for program counter samples
    - Calls _mcount_internal() for call arc samples
  - Calls _mcleanup() to end profiling
- hexconverter() – converts input data to correct format
  - Converts profiler data to correct format if necessary.
  - Based on format of serial monitor data capture.

gmon Handling

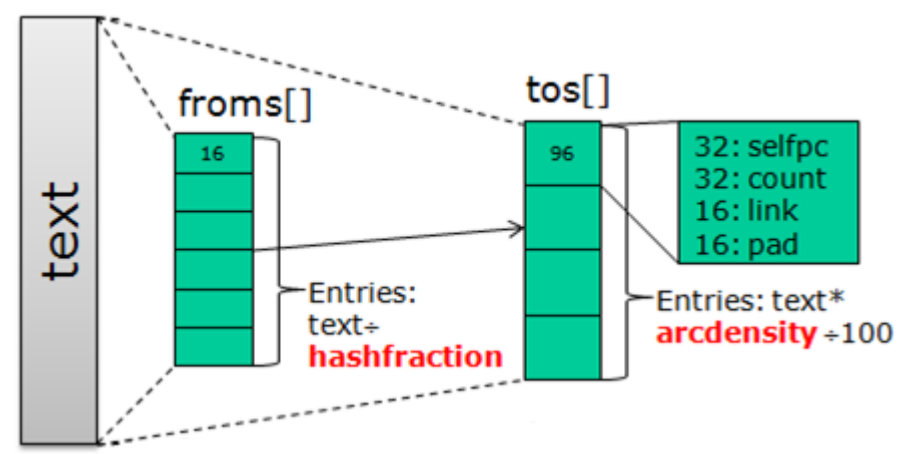
- moncontrol()
  - start/stop of profiling
- monstartup()
  - initialization of profiling
- _mcleanup()
  - cleanup function for profiler
- write_gmon()
  - writes profiler data to file
- write_hist()
  - writes pc sample histogram
- write_call_graph()
  - writes call graph arc samples




```
/*
 * The profiling data structures are housed in this structure.
 */
struct gmonparam {
 long int        state;      /* profiling state */
 unsigned short  *kcount;    /* array of PC sample counters */
 unsigned long   kcountsize; /* size of kcount[] array in bytes */
 ARCINDEX        *froms;     /* array of hashed from addresses*/
 unsigned long   fromssize;  /* size of froms[] array in bytes */
 struct tostruct *tos;       /* array of tos addresses with counter */
 unsigned long   tossize;    /* size of tos[] array in bytes */
 long            tolimit;    /* max number of tos[] elements */
 unsigned long   lowpc;      /* lower memory address bound */
 unsigned long   highpc;     /* upper memory address bound */
 unsigned long   textsize;   /* total memory size */
 unsigned long   hashfraction;     /* divider for froms[] hash */
 long            log_hashfraction; /* precomputed shift for hash */
};
```

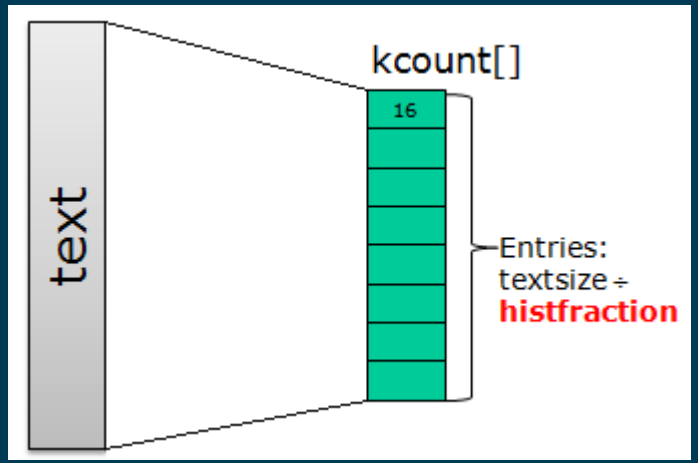
Call Graph Arc Processing

- _mcount_internal() – processes call arc samples
  - Converts caller address into froms[] array index.
  - froms[] contains indexes into tos[] array.
  - tos[] array stores callee address, counter, and a link to the next function in the call chain.



Program Counter Processing

- profil() – start/stop/configure statistical profiling.
  - Computes scaling factors to index into array.
- profil_count() – processing statistical profiling samples.
  - Converts memory address to array index.
  - Increments counter element at index.

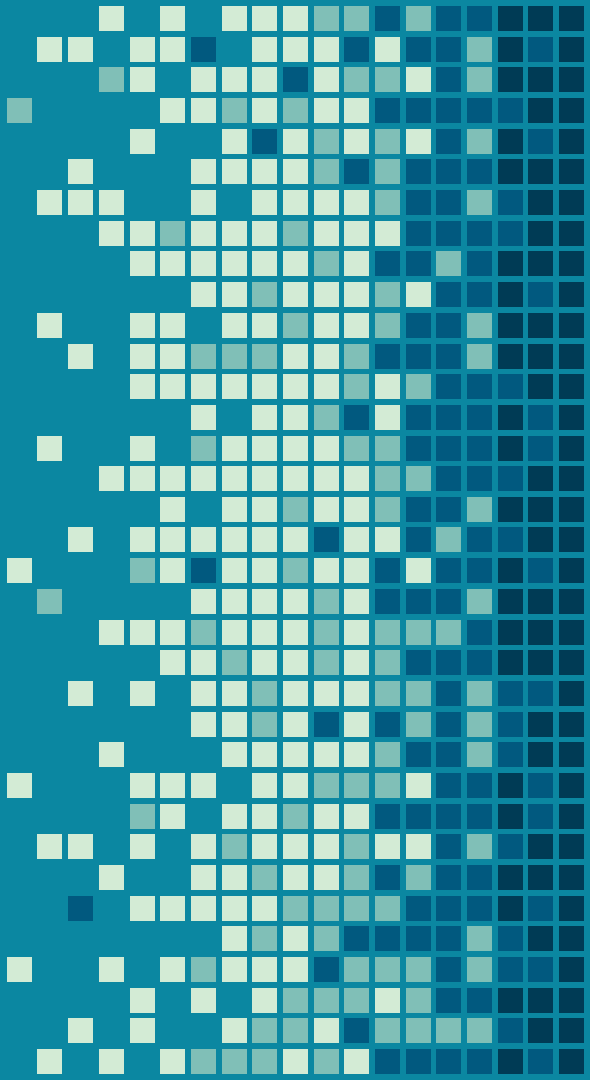


gprof

- gprof executable invoked from terminal.
- Requires target MCU executable and generated gmon.out.

```
C:\Users\Michael\Documents\GitHub\Lightweight-gprof-for-Microcontrollers\Results
\blinky>gprof blinky.elf gmon.out
Flat profile:

Each sample counts as 0.001 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 59.35     10.22    10.22                             loop
 18.93     13.48     3.26    13155     0.25     0.25  WAIT1_WaitCycles
  9.97     15.19     1.72                             WAIT1_Wait100Cycles
  9.51     16.83     1.64                             _mcount_internal
  1.16     17.03     0.20                             WAIT1_Wait10Cycles
  1.08     17.21     0.19    13155     0.01     0.26  WAIT1_WaitLongCycles
  0.01     17.21     0.00                             __gnu_mcount_nc
  0.00     17.21     0.00       26     0.00   132.46  WAIT1_Waitms
  0.00     17.21     0.00       26     0.00     0.00  control_LEDs
  0.00     17.21     0.00       14     0.00     0.00  BitIoLdd4_ClrVal
  0.00     17.21     0.00       13     0.00     0.00  BitIoLdd2_ClrVal
  0.00     17.21     0.00       13     0.00     0.00  BitIoLdd2_SetVal
  0.00     17.21     0.00       13     0.00     0.00  BitIoLdd3_ClrVal
  0.00     17.21     0.00       13     0.00     0.00  BitIoLdd3_SetVal
  0.00     17.21     0.00       12     0.00     0.00  BitIoLdd4_SetVal
```

DEMO



VALIDATION & ANALYSIS

Target MCU Validation

- Confirm retrieved values
- Confirm received values
- Mem values = variable values = transmitted values

r0	<-- top of stack
r1	
r2	
r3	
lr (callee address)	
lr (caller address)	<-- initial top of stack

Expression	Type	Value
(x)= sp	uint32_t	0x20002f78 (Hex)
(x)= sp + FRAME_SIZE + HW_RET_	unsigned int	0x20002fb0 (Hex)
(x)= pc	uint32_t	0x144c (Hex)

```
sp→ 0x20002F78  00000872  r...
    0x20002F7C  20002F78  x/.
    0x20002F80  20002F88  ./.
    0x20002F84  000009C5  Å...
    0x20002F88  00000000  ....
    0x20002F8C  00000000  ....
    0x20002F90  20002FB8  ./.
    0x20002F94  FFFFFFF9  ùÿÿÿ
    0x20002F98  0000BB80  .»..
    0x20002F9C  00000000  ....
    0x20002FA0  20002FBE  ¾/.
    0x20002FA4  20002FBE  ¾/.
    0x20002FA8  0000143D  =...
    0x20002FAC  00001449  I...
pc→ 0x20002FB0  0000144C  L...
```

```
sp→   0x20002FB0  0000BB80  .»..
      0x20002FB4  00000000  ....
      0x20002FB8  000000AC  ¬...
      0x20002FBC  0000BB80  .»..
to→   0x20002FC0  00001489  ....
from→ 0x20002FC4  000014E1  á...
```

Name	Value
MKL25Z128xxx4 (co	
r0	0x14e1
r1	0x1489
r2	0xac
r3	0xbb80
r4	0x0
r5	0x0
r6	0xffff
r7	0x20002fc8
r8	0xffffff80
r9	0x1
r10	0x1fff3000
r11	0x0
r12	0x1489
sp	0x20002fb0
lr	0x14e1 <WAIT1_Waitms+32>
pc	0x4da <__gnu_mcount_nc+10>
xpsr	0x21000000
msp	0x20002fb0
psp	0x200009a0
primask	0x0
basepri	0x0
faultmask	0x0
control	0x0

Host PC Validation

- Wrote Python script to verify host PC program.
  - Uses “pandas” library for easy/accurate processing.
  - Sums profiler samples for comparison to gprof profile.
- Secondary method to process the data to confirm results.
  - Offers confidence in original solution.
  - Provides easy way to validate changes.

pc.csv

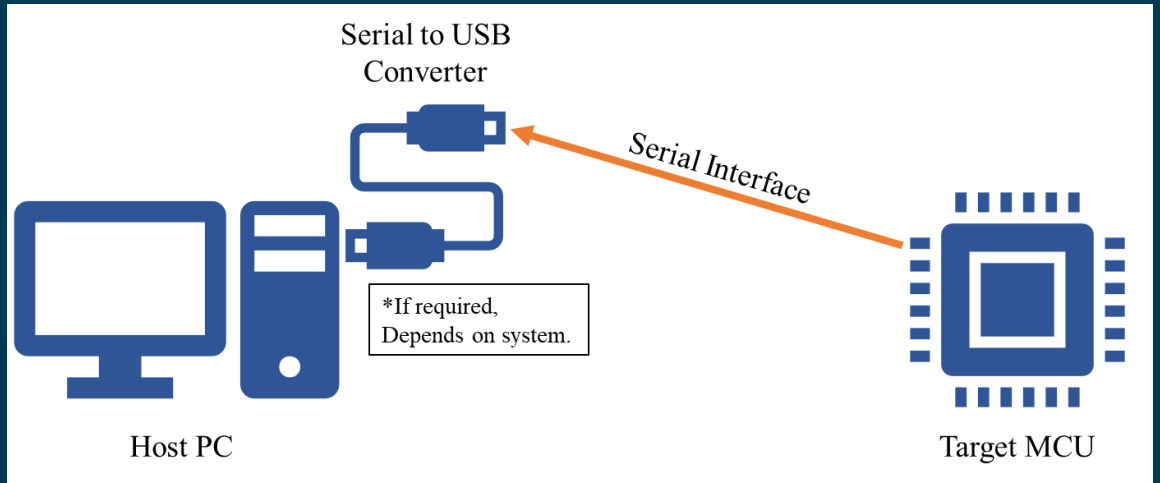
```
Addr1,Counter
0x4d6,1
0x7ce,177
0x7f6,2
0x81c,14
0x8a0,1444
0x1466,399
0x1468,44
0x146a,2988
0x146c,500
0x146e,1196
.
.
.
```

arc.csv

```
Addr1,Addr2,Counter
0x527,0xf21,14
0x533,0xf45,12
0x545,0xdc5,13
0x551,0xde9,13
0x563,0xe75,13
0x56f,0xe95,13
0x5bd,0x4fd,26
0x5c7,0x1529,26
0x150d,0x1499,13155
0x153d,0x14e5,13155
```

System Validation

- Write target MCU program with a known number of function calls and an established execution time.
- blinky program with static delays and finite number of cycles.
- Validates entire solution: target MCU application, serial transmission, host PC application, and gprof.



Overhead Analysis

Profiler	ROM Usage
Thesis - gprof	1340 B
Semihosting gprof	2000 B

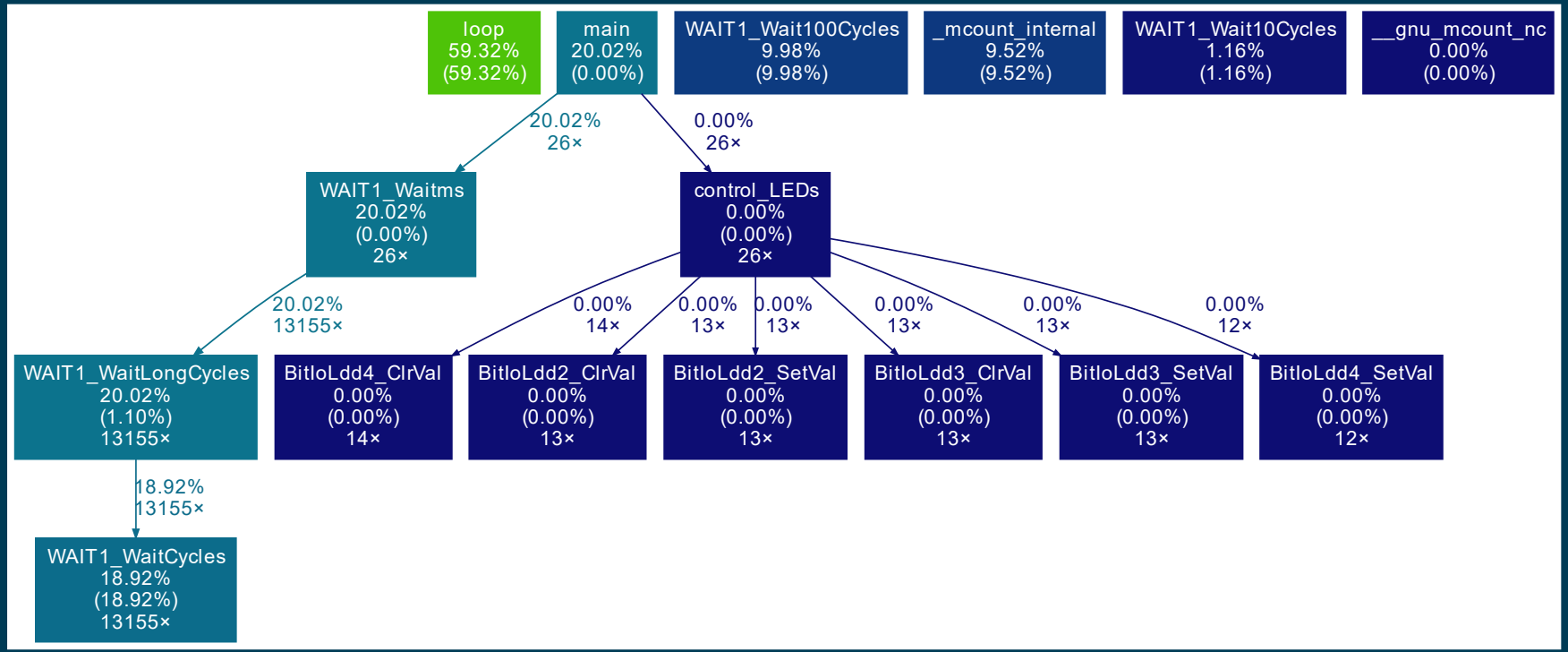
	Static Total RAM	Static Worst-Case RAM	Dynamic RAM
Thesis - gprof	246 B	206 B	0 B
Semihosting gprof	392 B	336 B	* 8.96kB for 4kB ROM

Description	Thesis - gprof Latency	Semihosting gprof Latency
Call Graph Arc Sample Processing Latency	29.24 us	6.21 us
Program Counter Sample Processing Latency	13.96 us	4.65 us
1 Byte Serial Transmission Latency	3.27 us	N/A
Debug Pin Control Latency	104.17 ns	104.17 ns
Context Switch Latency	312.50 ns	312.50 ns
Latency per Call Graph Arc Sample	**57.07 us**	**6.73 us**
Latency per Program Counter Sample	**30.08 us**	**5.28 us**

RESULTS

Blinky

- Flashes 8 states of an RGB LED changing state every 500ms.



Raw gprof Output

- Identical functionality to GNU gprof in a traditional setting.
- Provides the same performance and level of detail.

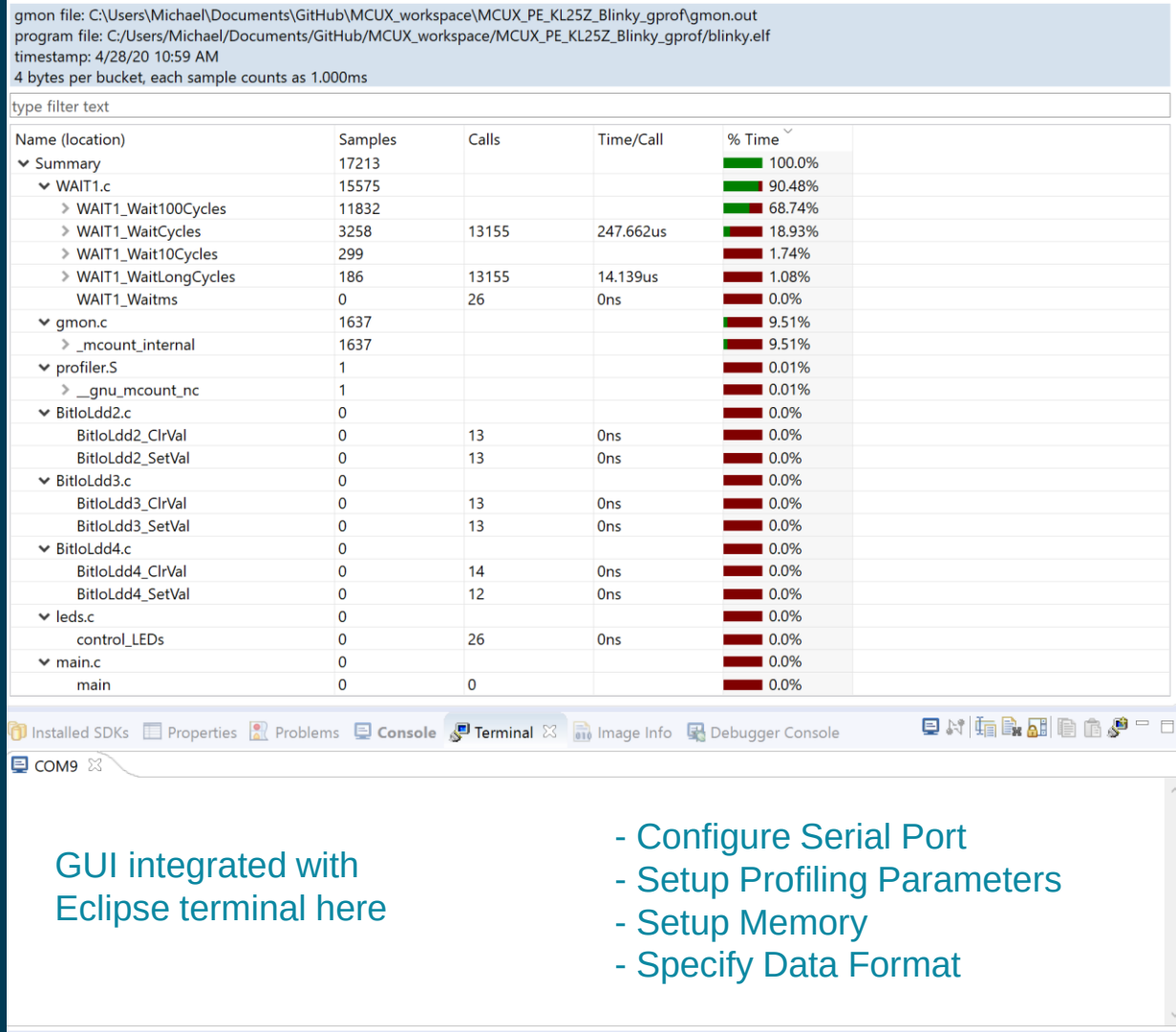
blink1gprof.txt

Many different results from validation runs are provided below:
https://github.com/mjdargen/Lightweight-gprof-for-Microcontrollers

FUTURE WORK

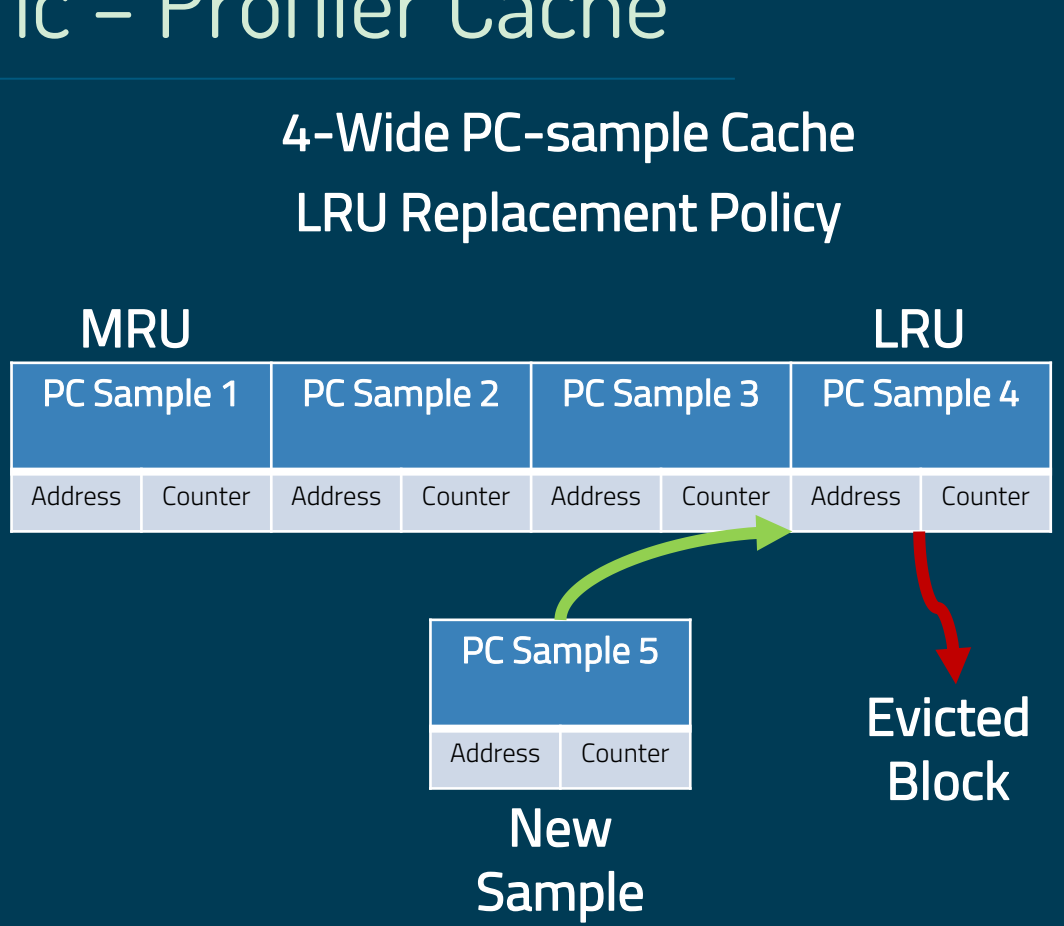
A More Automated Solution

- Integrate into Eclipse.
- Eclipse becoming the home for open-source MCU development.
- Easily GCC compilation.
- Integrate with serial terminal and existing gprof plugins.



Reduced Data Traffic – Profiler Cache

- Local cache to store call arcs and program counter samples
- Variable Cache Size
- Attempts to exploit localities
  - Temporal
  - Spatial
- Configurable Replacement Policies
  - Least Recently Used
  - Least Frequently Used
  - Least Proximal



Reduced Data Traffic – Indexed LUT

- Based on prior work by Dr. Dean.
- Local memory map look-up table.
- Contains start & end address of every function with an index.
- Requires multiple compilations.
  - Stabilize executable layout
  - Store memory map in source.
- Profiler transmits 1-byte index instead of a 4-byte mem address

```
#include "region.h"
const REGION_T RegionTable[] = {
  {0x00001cc1, 0x00001ce4, "LCD_24S_Write_Command"}, // 0
  {0x00001ce9, 0x00001d0c, "LCD_24S_Write_Data"}, // 1
  {0x000009d7, 0x00000b72, "_fp_digits"}, // 2
  {0x00000ded, 0x00000df6, "_printf_input_char"}, // 3
  {0x00001225, 0x00001410, "btod_internal_mul"}, // 4
  {0x00001411, 0x00001618, "btod_internal_div"}, // 5
  {0x000000c1, 0x000000c8, "__main"}, // 6
  {0x000000c9, 0x000000fc, "__scatterload_rt2"}, // 7
  {0x00000105, 0x0000011e, "__scatterload_copy"}, // 8
  {0x00000121, 0x0000013c, "__scatterload_zeroinit"}, // 9
  {0x000001a1, 0x000001ac, "Reset_Handler"}, // 10
  {0x000001ad, 0x000001ae, "NMI_Handler"}, // 11
  {0x000001af, 0x000001b0, "HardFault_Handler"}, // 12
  {0x000001b1, 0x000001b2, "SVC_Handler"}, // 13
  {0x000001b3, 0x000001b4, "PendSV_Handler"}, // 14
  {0x000001b5, 0x000001b6, "SysTick_Handler"}, // 15
  {0x000001e1, 0x00000204, "__2sprintf"}, // 16
  {0x00000209, 0x0000023a, "__2snprintf"}, // 17
  {0x00000241, 0x0000026c, "_printf_pre_padding"}, // 18
  {0x0000026d, 0x0000028e, "_printf_post_padding"}, // 19
  {0x0000028f, 0x000002e0, "_printf_str"}, // 20
  {0x000002e1, 0x0000033a, "_printf_int_dec"}, // 21
  {0x0000034d, 0x000003ec, "strcmp"}, // 22
```

Conclusion

Problem

- MCUs are constrained…
- therefore, profile-guided optimization is critical…
- but existing profilers exhaust limited resources…
- therefore, a new and less intensive profiler must be developed.

Solution

- Low memory footprint
- Lightweight - distributed workload
- Open-source - gprof-based
- MCU-agnostic
- Provides identical functionality to gold standard profiler.
- Scalable framework