

PROFino

- **ALUNOS:**

José Vitor Santos Silva

Wesley Alves Santana

Introdução

Partição

function	calls	time (s)	time (%)
main	1	0.028672	1.182432
particiona	10	0.380927	15.70945
quickSort	21	1.974271	81.41891
total execution time (s) 2.424832			

GUI

≡

PROFino

— ×


USB Port:

/dev/ttyACM0

Build and Upload

Start Profiling

Select File:



quicksort.c

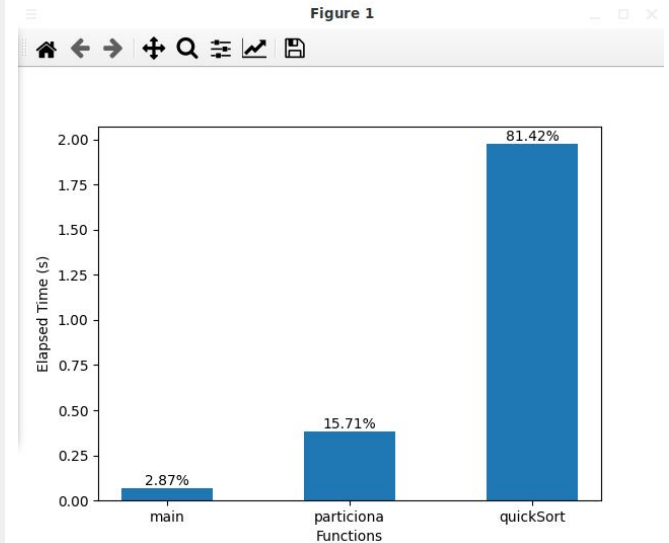
Stop Profiling

Logs:

function	calls	time (s)	time (%)
main	1	0.069632	2.871621
particiona	10	0.380927	15.70945
quickSort	21	1.974271	81.41891

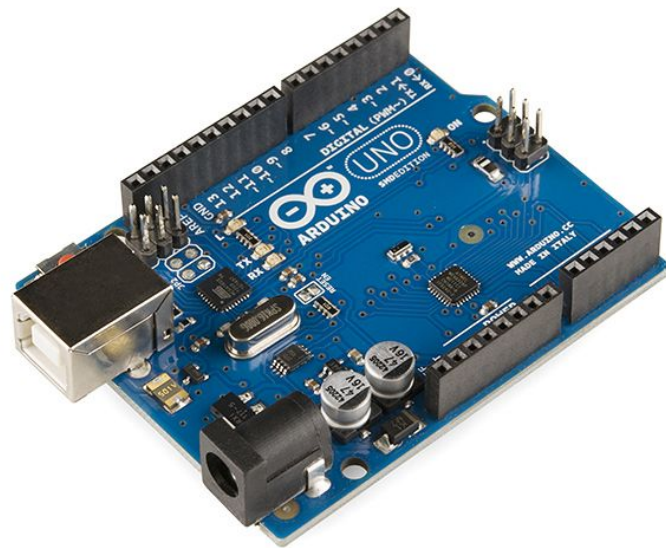
total execution time (s) 2.424832

program finished

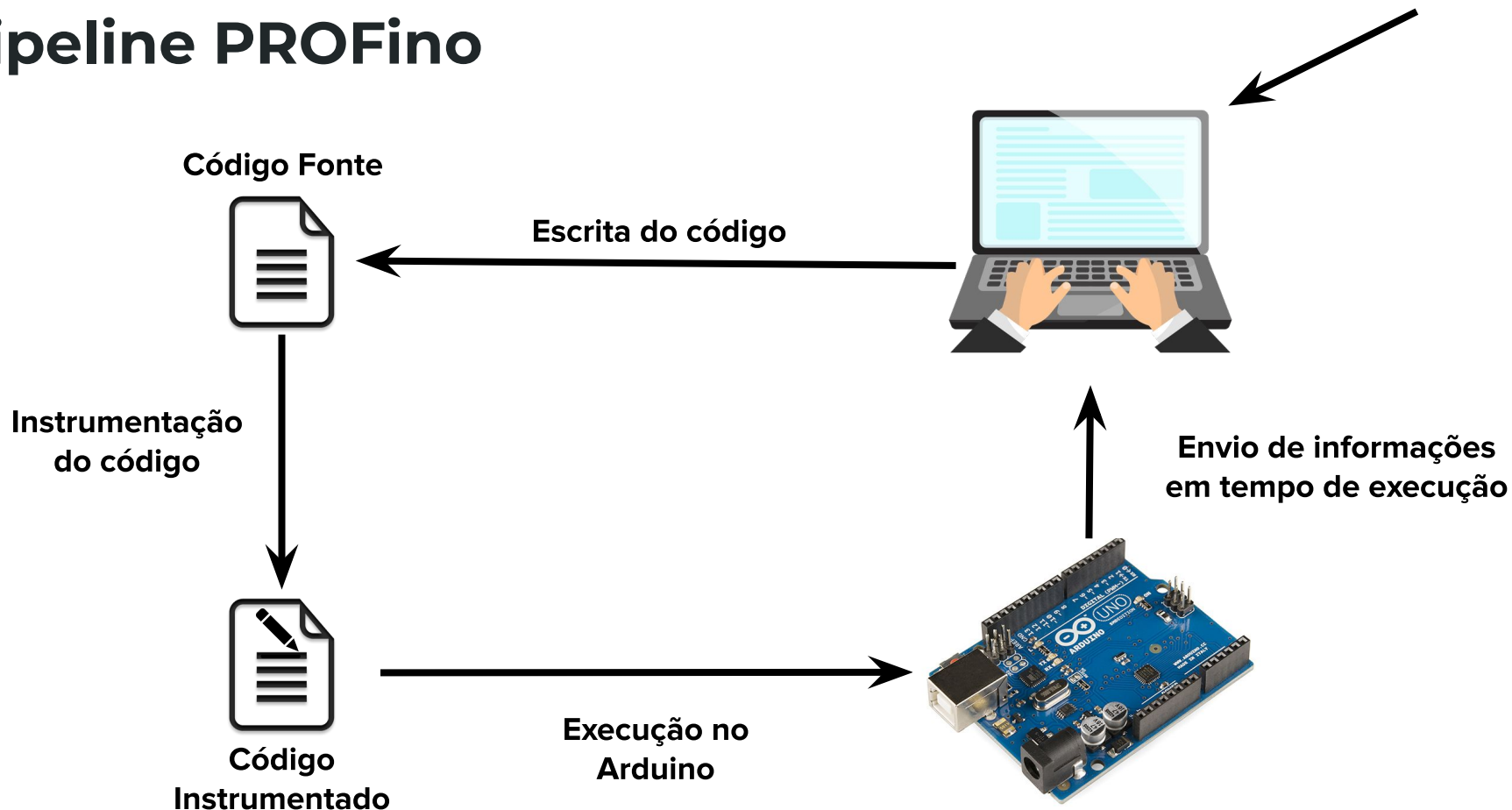


Motivação

- Entender como o programa está se comportando;
- Saber o que gasta mais tempo;
- Dificuldade em obter informações da execução de programas embarcados;
- Obter métricas para auxiliar a otimização do programa;



Pipeline PROFino



Detalhes de implementação

- Comunicação Arduino-Host
 - Comunicação Serial via UART

```
15 void uart_init(void) {
16     UBRR0H = UBRRH_VALUE;
17     UBRR0L = UBRL_VALUE;
18
19     #if USE_2X
20         UCSR0A |= _BV(U2X0);
21     #else
22         UCSR0A &= ~(_BV(U2X0));
23     #endif
24     UCSR0C = _BV(UCSZ01) | _BV(UCSZ00);
25     UCSR0B = _BV(RXEN0) | _BV(TXEN0);
26 }
```

```
28 void uart_putchar(char c, FILE *stream) {
29     if (c == '\n') {
30         uart_putchar('\r', stream);
31     }
32     loop_until_bit_is_set(UCSR0A, UDRE0);
33     UDR0 = c;
34 }
```

Detalhes de implementação

- Instrumentação
 - ctags-universal
- Coleta dos dados de tempo de execução
 - Utilização do Timer0 do Arduino Timer/Counter Register – TCNT0
 - Timer controlado pelo Timer/Counter Control Register – TCCR0

TCCR0:

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CS = Clock select

Prescalers para mudar o clock do timer

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$\text{clk}_{\text{I/O}}$ /(No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

F_CPU = 16 MHz

Prescaler	Clock Frequency	Timer Count
8	2 MHz	15999
64	250 kHz	1999
256	62.5 kHz	499
1024	15625 Hz	124

Tempo gasto para timer0 dar overflow = $1/62500 * 256 = 4,096$ ms

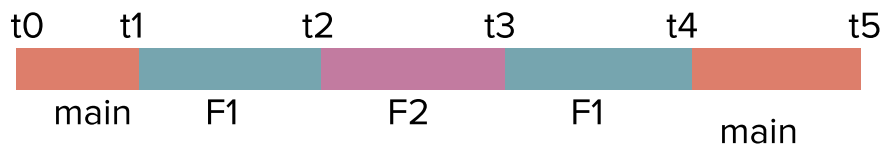
Coleta do tempo utilizando interrupção do overflow do timer

```
7  volatile uint16_t timer_overflow_count;
8  volatile uint16_t timer_overflow_count_overflow;
9
10 void timer0_init(){
11     TCCR0B |= (1 << CS02);
12     TCNT0 = 0;
13     TIMSK0 |= (1 << TOIE0);
14     sei();
15     timer_overflow_count = 0;
16     timer_overflow_count_overflow = 0;
17 }
```

```
19 ISR(TIMER0_OVF_vect){
20     timer_overflow_count++;
21     if (timer_overflow_count ≥ 32000){
22         timer_overflow_count_overflow++;
23         timer_overflow_count = 0;
24     }
25 }
```

Contabilização do tempo gasto em cada função

```
1 void F2(){
2     //code
3 }
4 void F1(){
5     // code
6     F2();
7     //code
8 }
9 int main(){
10     F1();
11     return 0;
12 }
```



Função	Chamadas	Tempo
main	0	$(t_1 - t_0) + (t_5 - t_4)$
F1	0	$(t_2 - t_1) + (t_4 - t_3)$
F2	0	$(t_3 - t_2)$

- Quando entra em uma função incrementa o tempo da anterior.
- Quando sai incrementa o da que saiu.

Pilha:

(F2, t_2)

(F1, t_3)

(main, t_0)

Desafios encontrados

- Comunicação serial
 - Algumas implementações testadas não funcionaram de maneira apropriada
- Instrumentação
 - Dificuldade em como encontrar o início e fim das funções;
 - Tentativa de usar diretiva de compilação usada para instrumentação (-finstrument-functions);
 - Testamos algumas bibliotecas de parser para python (pycparser) mas que não funcionaram.
- Temporização da execução
 - Inicialmente não sabíamos como medir o tempo no Arduino, a ausência de um RTC e a sugestão do professor fez com que usássemos timers.
- Testes
 - A otimização de espaço utilizada na compilação (-Os) fez alguns códigos de teste se comportarem de modo diferente no Arduino, quando comparados com o gprof e gcov, isso causou algumas dúvidas..

Demonstração

- Quicksort

```
28  int main(){
29      srand(23);
30      int arr[TAM] = {6, 3, 17, 54, 33, 12, 78, 1, 4, 22, 56, 34, 2, 65, 43};
31      quickSort(arr, 0, TAM-1);
32      return 0;
33  }
```

```
21  void quickSort(int arr[], int ini, int fim){
22      if(ini < fim){
23          int pivot = particiona(arr, ini, fim);
24          quickSort(arr, ini, pivot-1);
25          quickSort(arr, pivot+1, fim);
26      }
27  }
```

```

4  int particiona(int arr[],int ini, int fim){
5      int pivot = arr[fim], i=ini, j=fim;
6      while(i<j){
7          while(arr[i] < pivot)
8              i++;
9          while(arr[j] ≥ pivot && j ≥ ini)
10             j--;
11         if(i<j){
12             int temp;
13             temp = arr[i];
14             arr[i]=arr[j];
15             arr[j]=temp;
16         }
17     }
18     arr[fim]=arr[i]; arr[i]=pivot;
19     return i;
20 }

```

GPROF

→ PROFino git:(main) X gprof quicksort.bin

Flat profile:

Each sample counts as 0.01 seconds.
no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	10	0.00	0.00	particiona

PROFino

function	calls	time (s)	time (%)
main	1	0.028672	1.182432
particiona	10	0.380927	15.70945
quickSort	21	1.974271	81.41891
total execution time (s) 2.424832			

GCOV

```
11: 21: void quickSort(int arr[], int ini, int fim){
21: 22:     if(ini < fim){
10: 23:         int pivot = particiona(arr, ini, fim);
10: 24:         quickSort(arr, ini, pivot-1);
10: 25:         quickSort(arr, pivot+1, fim);
-: 26:     }
11: 27: }
1: 28: int main(){
1: 29:     srand(23);
1: 30:     int arr[TAM] = {6, 3, 17, 54, 33, 12, 78, 1, 4, 22, 56, 34, 2, 65, 43};
1: 31:     quickSort(arr, 0, TAM-1);
1: 32:     return 0;
-: 33: }
```