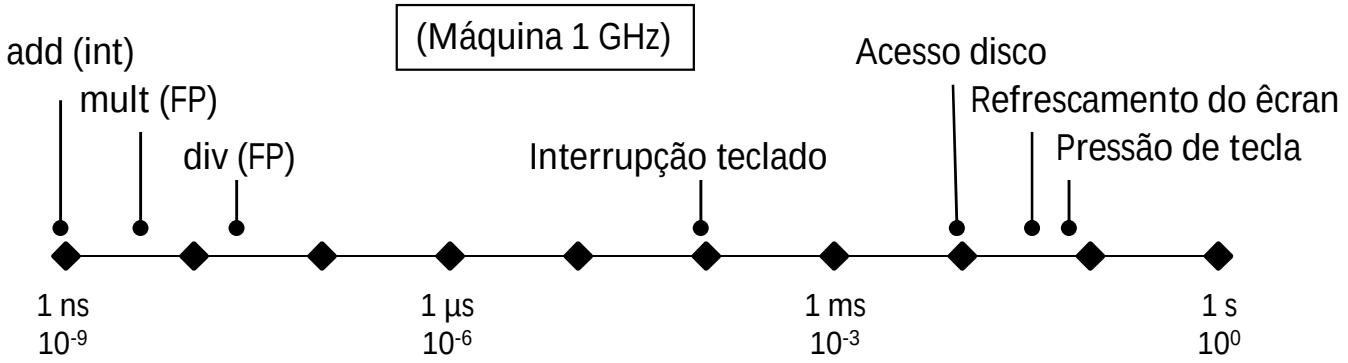
Medição e documentação do desempenho

Luís Paulo Peixoto dos Santos

Maio, 2010

Tempo de Execução de um Segmento de Código

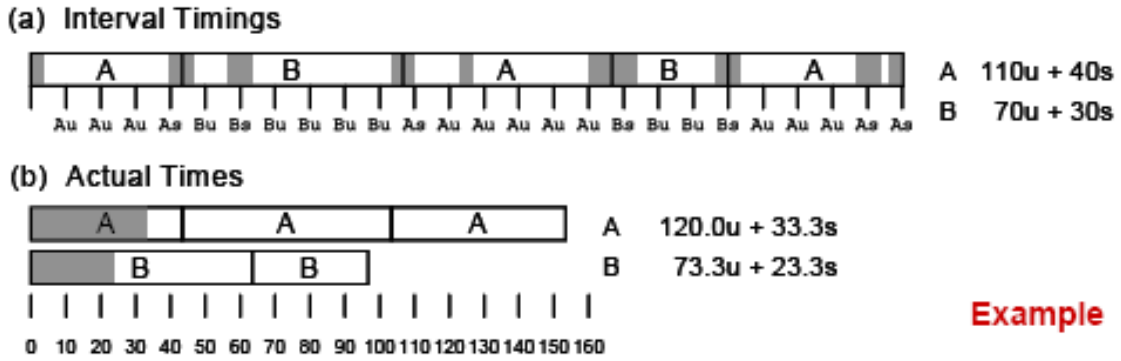
```
before = ReadTimer();
<<< Code Segment>>>
after = ReadTimer();
ElapsedTime = after - before;
```



- Resolução do relógio: unidade de tempo entre 2 incrementos do contador
  não é possível medir eventos com duração inferior à resolução
- Precisão do relógio: diferença entre o valor medido e o tempo efectivamente decorrido

Tempo de Execução

- O sistema operativo (UNIX) mantém 2 contadores por processo para contabilizar *user* e *system times*



- A contagem de tempo é feita por contagem de intervalos: regularmente o SO é interrompido (período típico entre 1 a 10 ms) e atribui a última *time slice* ao processo que estiver a executar naquele momento
- Usar apenas para medição superiores a 1 s!

`time` (Unix)

SYNOPSIS

time [*options*] *command* [*arguments...*]

DESCRIPTION

The time command runs the specified program *command* with the given arguments. When *command* finishes, time writes a message to standard error giving timing statistics about this program run. These statistics consist of (i) the elapsed real time between invocation and termination, (ii) the user CPU time and (iii) the system CPU time

EXAMPLE

```
time ls
real  0m0.035 s
user  0m0.000 s
sys   0m0.003 s
```

times (Unix)

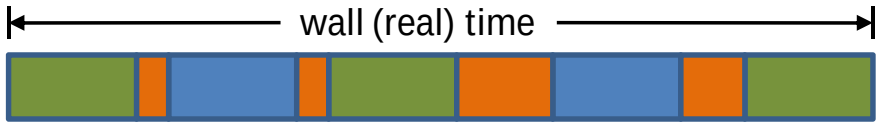
- Os timers do Sistema Operativo podem ser lidos pelo programa:

```
#include <sys/times.h>
struct tms {
  clock_t tms_utime;   // user time
  clock_t tms_stime;   // system time
  clock_t tms_cutime;  // user children time
  clock_t tms_cstime;  // system children time
};

clock_t times (struct tms *buf);
```

Tempo de Execução

- *Wall Time* = *user time* + *system time* + *other processes time*



User time: tempo a executar instruções do processo do utilizador

System time: tempo a executar instruções do *kernel* em nome do utilizador

Other processes time: tempo a executar instruções de outros processos

- *Wall time*
  - é o tempo decorrido desde o início ao fim da execução do programa
  - dependente da carga do sistema (*other processes time*)
  - *System time* corresponde essencialmente a *system calls* e tratamento de interrupções e excepções

gettimeofday (Unix)

```
#include <sys/time.h>
struct timeval {
  long tv_sec; /* seconds */
  long tv_usec; /* microseconds */
};

int gettimeofday(struct timeval *tv, struct timezone *tz);
```

Devolve em tv os segundos e microsegundos decorridos desde algum instante referência. `tz` não é usado- passar `NULL` como parâmetro.

Sistema	Resolução (µs)
Pentium II, Windows NT	10.000
Compaq Alpha	977
Pentium III, Linux	1
Sun UltraSparc	2

Contador de ciclos (`rdtsc`)

- Contador incrementado a cada ciclo do relógio do processador
  - resolução de 0.5 ns numa máquina a 2GHz
- Lido com a instrução *assembly* `rdtsc`, que coloca os 32 bits mais significativos no `%edx` e os 32 menos significativos no `%eax`
- Ferramenta poderosa para medição do wall time com grande resolução

Contador de ciclos (`rdtsc`)

```
static unsigned cyc_hi = 0, cyc_lo = 0;

static void access_counter(unsigned *hi, unsigned *lo) {
 asm("rdtsc; movl %%edx,%0; movl %%eax,%1" /* Read cycle counter */
 : "=r" (*hi), "=r" (*lo) /* and move results to */
 : /* No input */ /* the two outputs */
 : "%edx", "%eax"); }

void start_counter() {
 access_counter(&cyc_hi, &cyc_lo); }

double get_counter() {
 unsigned ncyc_hi, ncyc_lo, hi, lo, borrow;
 double result;

 access_counter(&ncyc_hi, &ncyc_lo);

 lo = ncyc_lo - cyc_lo;
 borrow = lo > ncyc_lo;
 hi = ncyc_hi - cyc_hi - borrow;
 result = (double) hi * (1 << 30) * 4 + lo;
 return result; }
```

Contador de ciclos (`rdtsc`)

```
/* Estimate the clock rate by measuring the cycles that elapse */
/* while sleeping for sleeptime seconds */

double mhz(int verbose, int sleeptime) {
 double rate;
 start_counter();
 sleep(sleeptime);
 rate = get_counter() / (1e6*sleeptime);
 if (verbose)
  printf("Processor clock rate ~= %.1f MHz\n", rate);
 return rate;
}
```

Contador de ciclos (`rdtsc`)

- Em máquinas com múltiplos processadores (Symmetric Multi Processors, multi core) duas leituras distintas com `rdtsc` podem devolver valores não relacionados:
  - Se entre a 1ª e a 2ª leitura o processo migra de um processador para outro , então são lidos diferentes contadores
  - Os contadores dos diferentes processadores não são sincronizados, logo as duas leituras não fazem sentido entre si
  - Uma solução é forçar o processo ou *thread* a executar sempre no mesmo processador (*process/thread affinity*)
- Em máquinas com frequência de relógio variável `rdtsc` é bem mais complexo de usar!

Resolução

- Duração do evento a medir da mesma ordem de grandeza da resolução do relógio
  - Cada medição deve consistir em N repetições do código
  - O valor da medição é o tempo decorrido sobre o número de repetições

```
before = ReadTimer();
for (i=0 ; i< N ; i++) {
        <<< Code Segment>>>
}
after = ReadTimer();
ElapsedTime = (after – before)/N;
```

Variação nas medições

- O tempo medido varia entre medições:
  - carga da máquina
  - hierarquia de memória
  - *pipelining* e *branch prediction*

- Realizar várias medições e apresentar uma estatística
  - Média: sensível a *outliers* ; apresentar também desvio padrão
  - Mínimo: valor obtido em condições ideais
  - Mediana: menos sensível a variações

Variação nas medições: *cache*

- O tempo de execução de um segmento de código pode depender do facto de os dados e instruções utilizados se encontrarem ou não nos níveis mais elevados da hierarquia

- Dependendo dos objectivos:
  - fazer o *warm up* da *cache*
    - executar uma vez o segmento de código sem medir
  - limpar a cache
    - executar outro segmento de código que aceda a um conjunto de dados diferente e suficientemente extenso para limpar a *cache*

Variação nas medições

```
<<< Code Segment>>> // cache warm up
for (j=0 ; j<M ; j++) {
  before = ReadTimer();
  for (i=0 ; i< N ; i++) {
        <<< Code Segment>>>
  }
  after = ReadTimer();
  ElapsedTime[j] = (after – before)/N;
}
ComputeStat (ElapsedTime, M);
```

Apresentação de Resultados

- Documentar as condições experimentais
- Documentar métricas e respectivas unidades, estatísticas e metodologia de análise de resultados
- Apresentar resultados de forma compacta
  - Apresentar tabelas e gráficos
- Analisar resultados obtidos
  - justificar valores obtidos (com ênfase nos valores fora do intervalo típico)
  - extrair conclusões de alto nível, eventualmente suportadas por modelos matemáticos

Condições experimentais

- Intel Pentium Core 2 Duo E7500 2.93 GHz (L2 3 MB, 1066 MHz FSB)
- 4Gb DDRII 800 Mhz Kingston
- Mother Board ASUS P5KPL-AM/SE
- 500Gb SATAII 7200 Rpm 16MB Cache
- Windows 7 Professional

Metodologia experimental

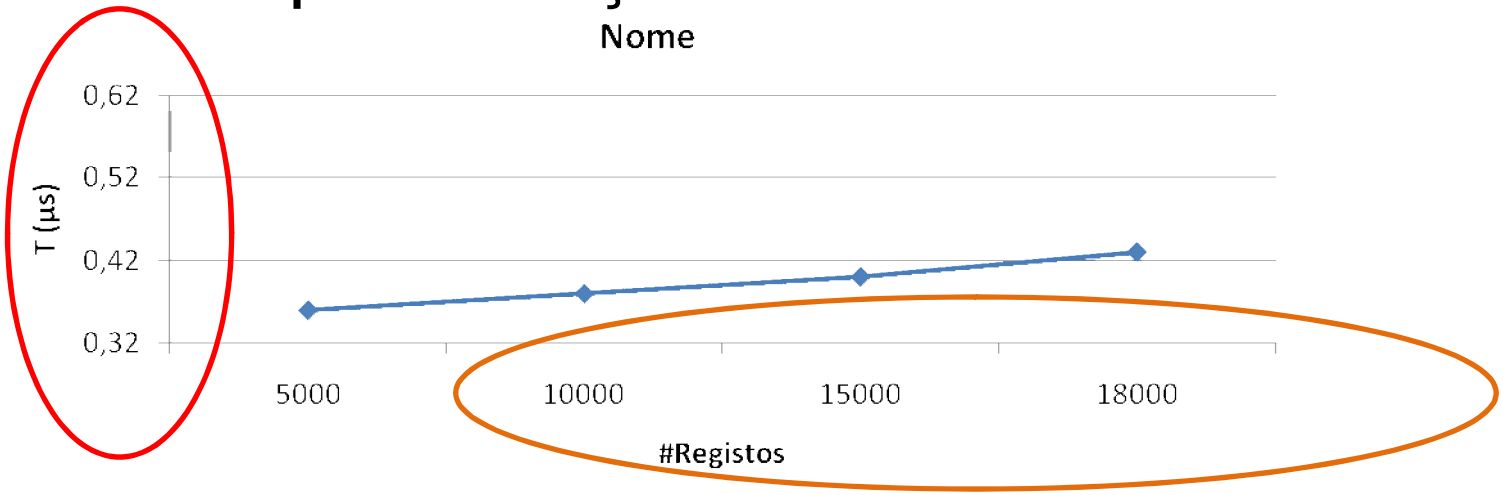
- Tempo de inserção de 1 registo reportado em µs
- Medições baseadas no relógio do processador (rdtsc) com resolução de 1/2.93 GHz=0.34 ns
- 1 medição consiste na média de 100 operações (N=100)
- Efectuam-se 20 medições (M=20)
- Reporta-se a mediana das medições efectuadas
- Medições efectuadas para base de dados com 5000, 10000, 15000 e 18000 registos

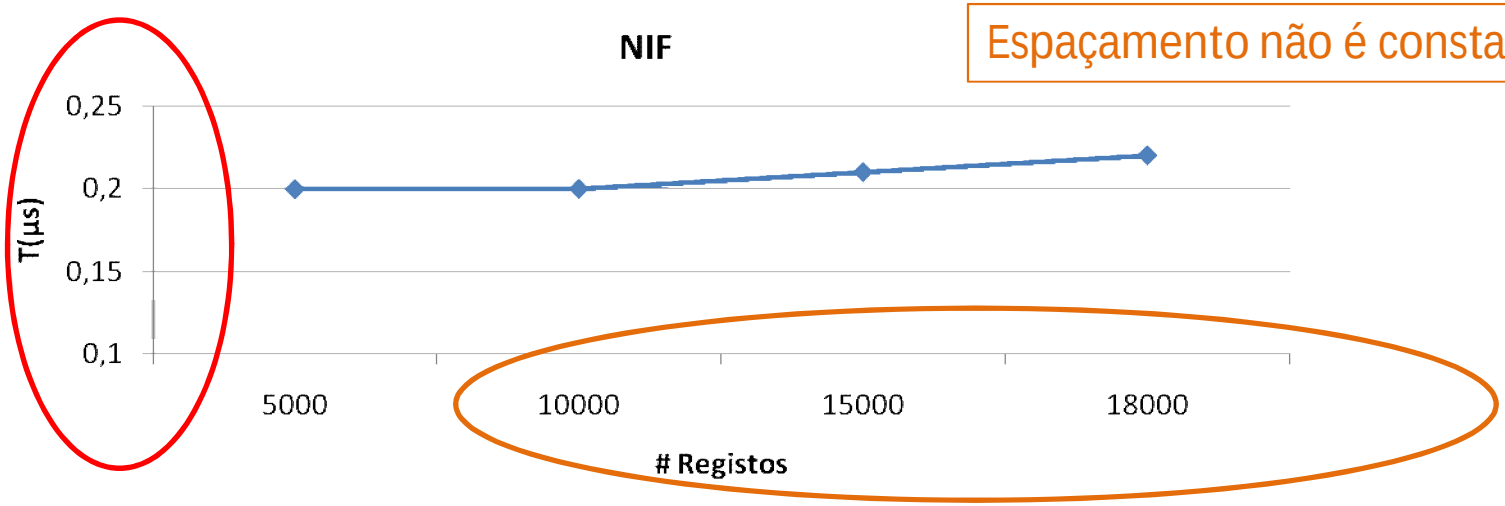
Apresentação de Resultados

Tempo de Execução (µs)				
	Nº de registos			
Operação	5000	10000	15000	18000
Carregar Dados	10 019.00	20 881.00	32 027.00	40 992.00
Inserir User	7.10	7.40	8.80	9.50
Procura (Nome)	0.36	0.38	0.40	0.43
Procura (NIF)	0.20	0.20	0.21	0.22
Percorrer Estrutura	92.00	232.00	470.00	673.00

Tabela 1 – Tempos de execução (µs) para diversos tamanhos da base de dados

Apresentação de Resultados

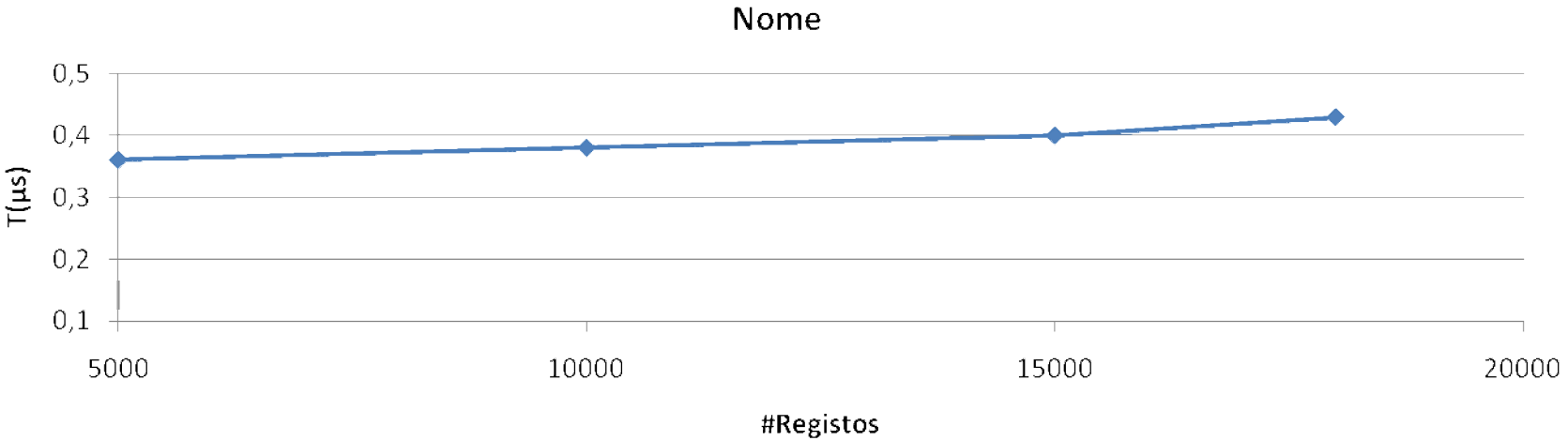


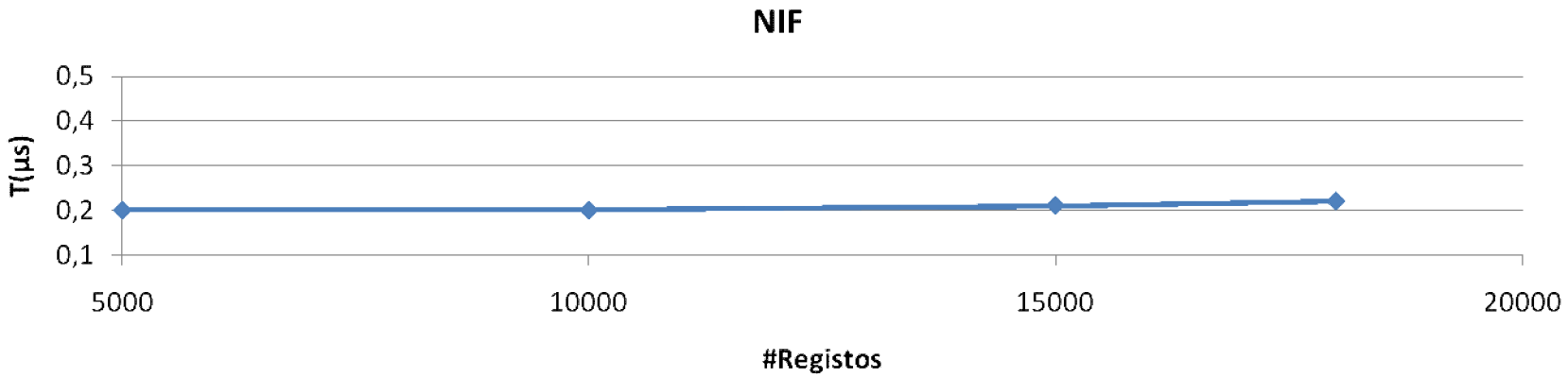


Espaçamento não é constante!

Escalas diferentes em comparações directas induzem em erro!

Apresentação de Resultados





Apresentação de Resultados