Programação em Sistemas Computacionais

Avaliação e optimização de desempenho

Centro de Cálculo
Instituto Superior de Engenharia de Lisboa

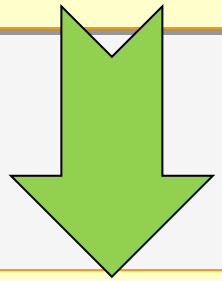
Pedro Pereira palex@cc.isel.ipl.pt

Limitações das optimizações do compilador

side effect

```
void fx(int x) {
  printf("res=%d",f(x)+f(x)+f(x));
}
```

res= ?



```
fx(1);
```

```
int f(int x) {
  return x + 1;
}
```

```
void fx(int x) {
 printf("res=%d",3*f(x));
}
```

res= ?

```
int f(int x) {
  static int i=1;
  return x + i++;
}
```

Optimizar se:

a função não tem efeitos colaterais, ou seja,

apenas retorna um valor que depende dos argumentos

Limitações das optimizações do compilador

memory aliasing

```
void fx(int *xp, int *yp) {
  *xp += *yp;
  *xp += *yp;
}
```

```
mov  edx,[ebp+8]
mov  ecx,[ebp+12]
mov  eax,[ecx]
add  [edx],eax
mov  eax,[ecx]
add  [edx],eax
```

```
void fx(int *xp, int *yp) {
  *xp += 2 * *yp;
}
```

```
mov  edx,[ebp+8]
mov  ecx,[ebp+12]
mov  eax,[ecx]
add  eax, eax
add  [edx], eax
```

```
int x, y;
x = 2; y=1;
fx( &x, &y);
```

x = ?

```
int x;
x = 2;
fx( &x, &x);
```

x = ?

Optimizar se:

o ponteiro xp nunca apontar o mesmo que yp

Avaliação de desempenho – funções para medir tempo

timer.h

```
void start_timer();
long get_microtime();
```

usa

```
#include <time.h>

int clock_gettime(clockid_t id, struct timespec *tp);
```

ttest.c

```
#include <stdio.h>
#include "timer.h"

int main() {
  long tm;
  puts("Hit enter.");
  start_timer();
  getchar();
  tm = get_microtime();
  printf("%ld micro seconds.\n",tm);
  return 0;
}
```

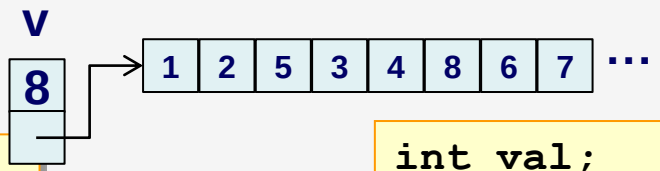
gcc –Wall –lrt –o ttest ttest.c timer.o

Vector - exemplo para avaliar desempenho

vector.h

```
struct _vector {
  int len;
  int *vec;
};
typedef struct _vector Vector;

void init_vector(Vector *v, int size, int *array);
int length_vector(Vector *v);

int elem_vector(Vector *v, int idx, int *dest);
```

Funções implementadas em vector.c




```
Vector v; int a[MAX];
…
init_vector(&v,8,a);
```

```
int val;
if (elem_vector(&v,5,&val))
  printf("v[5]=%d\n",val);
else
  puts("Index out of bounds.");
```

```
printf("v.length=%d\n",length_vector(&v));
```

Função avaliada – soma todos os elementos do vector

svector1.c

```
#include "vector.h"

void sum(Vector *v, int *dest) {
  int i, val;
  *dest = 0;
  for (i = 0; i < length_vector(v); i++) {
    elem_vector(v, i, &val);
    *dest += val;
  }
}
```

t1_svector.c

```
void sum(Vector *v, int *res);

int main() {
  Vector v; int elems[MAX];
  long tm; int res, r;
  init_vector(&v,8*1024,elems);
  for(r=0 ; r<RUNS ; ++r) {
    start_timer();
    sum(&v,&res);
    tm=get_microtime();
    printf("sum=%d em %ld msecs.\n",
           res,tm);
  }
  return 0;
}
```

Função avaliada – exemplo de resultados

t1_svector.c

```
void sum(Vector *v, int *res);

int main() {
  Vector v; int elems[MAX];
  long tm; int res, r;
  init_vector(&v,8*1024,elems);
  for(r=0 ; r<RUNS ; ++r) {
    start_timer();
    sum(&v,&res);
    tm=get_microtime();
    printf("sum=%d em %ld msecs.\n",
           res,tm);
  }
  return 0;
}
```

```
sum=-1265659096 em 354 msecs.
sum=-1265659096 em 362 msecs.
sum=-1265659096 em 274 msecs.
sum=-1265659096 em 323 msecs.
sum=-1265659096 em 26 msecs.
sum=-1265659096 em 275 msecs.
sum=-1265659096 em 274 msecs.
sum=-1265659096 em 304 msecs.
sum=-1265659096 em 275 msecs.
sum=-1265659096 em 274 msecs.
sum=-1265659096 em 301 msecs.
sum=-1265659096 em 274 msecs.
sum=-1265659096 em 275 msecs.
sum=-1265659096 em 275 msecs.
sum=-1265659096 em 0 msecs.
sum=-1265659096 em 275 msecs.
sum=-1265659096 em 274 msecs.
sum=-1265659096 em 12244 msecs.
sum=-1265659096 em 276 msecs.
```

??

??

Como medir o desempenho?

- **O sistema operativo pode colocar outro processo a executar durante a avaliação**
  - Executar N vezes e escolher o menor tempo (que se repete várias vezes)
- **A máquina virtual (VMware) acerta o relógio (ocasionalmente) pelo sistema *host***
  - Executar N vezes e dispensar os tempos demasiado baixos.

- **Uma solução geral:**
  - Executar N vezes, ordenar crescentemente os tempos obtidos e escolher o valor em N/4.

Função avaliada – exemplo de resultados filtrados

t2_svector.c

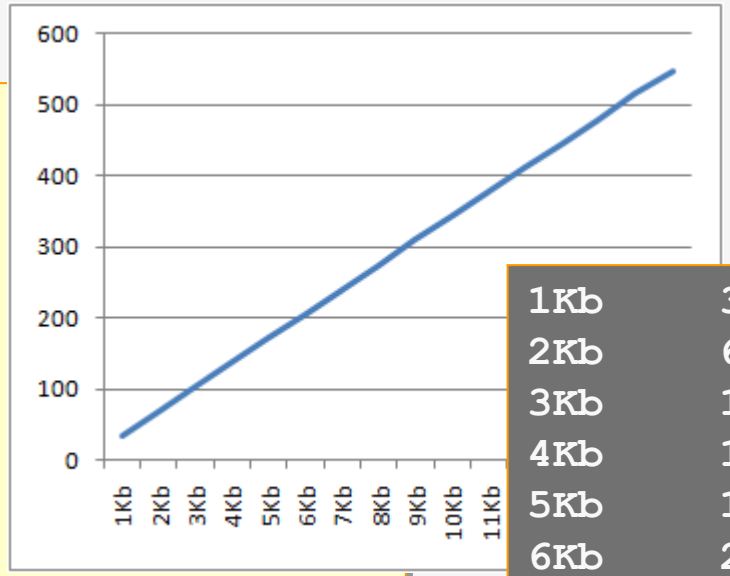
```
void sum(Vector *v, int *res);

long times[RUNS];

int main() {
  Vector v; int elems[MAX];
  long tm; int res, r;
  init_vector(&v,8*1024,elems);
  for(r=0 ; r<RUNS ; ++r) {
    start_timer();
    sum(&v,&res);
    times[r]=get_microtime();
  }
  sortTimes();
  for(r=0 ; r<RUNS ; ++r) printf("%ld ",times[r]);
  printf("\n%ld msecs.\n",times[RUNS/4]);
  return 0;
}
```

```
54 62 274 274 274 274 274 274
274 274 274 274 274 274 274 275
275 275 275 275 301 302 303 306
309 334 344 346 359 389 431 1661
274 msecs.
```

Função avaliada – resultados para várias dimensões

t3_svector.c

```
void sum(Vector *v, int *res);

long times[RUNS];

int main() {
  Vector v; int elems[16*KB];
  int sz, run, res;
  for(sz=KB ; sz<=16*KB ; sz+=KB) {
    init_vector(&v,sz,elems);
    for(run=0 ; run<RUNS ; ++run) {
      start_timer();
      sum(&v,&res);
      times[run]=get_microtime();
    }
    sortTimes();
    printf("%dKb\t%ld\n",sz/KB,times[RUNS/4]);
  }
  return 0;
}
```

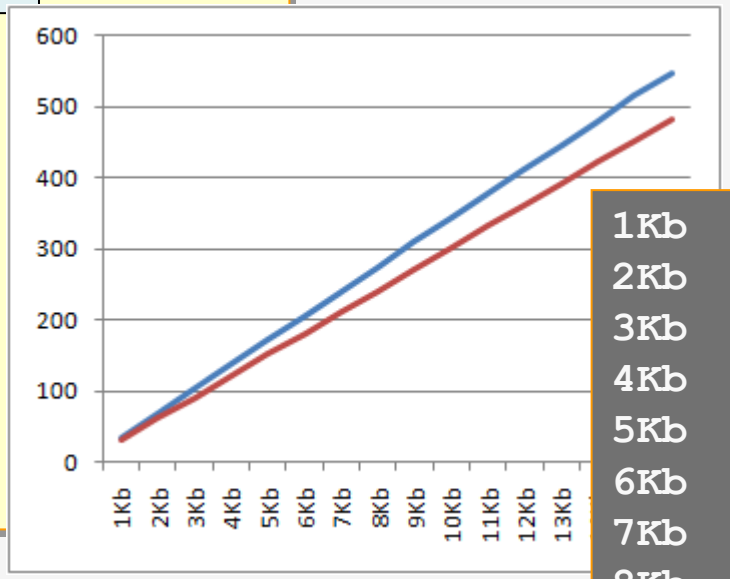


1Kb	35
2Kb	69
3Kb	103
4Kb	137
5Kb	171
6Kb	206
7Kb	240
8Kb	274
9Kb	308
10Kb	343
11Kb	377
12Kb	411
13Kb	445
14Kb	480
15Kb	514
16Kb	548

Optimizar a função sum – versão 2

svector2.c

```
#include "vector.h"

void sum(Vector *v, int *dest) {
  int i, val, len;
  *dest = 0;
  len = length_vector(v);
  for (i = 0; i < len; i++) {
    elem_vector(v, i, &val);
    *dest += val;
  }
}
```



1Kb	31
2Kb	61
3Kb	91
4Kb	121
5Kb	151
6Kb	181
7Kb	211
8Kb	241
9Kb	272
10Kb	301
11Kb	331
12Kb	361
13Kb	391
14Kb	421
15Kb	452
16Kb	482

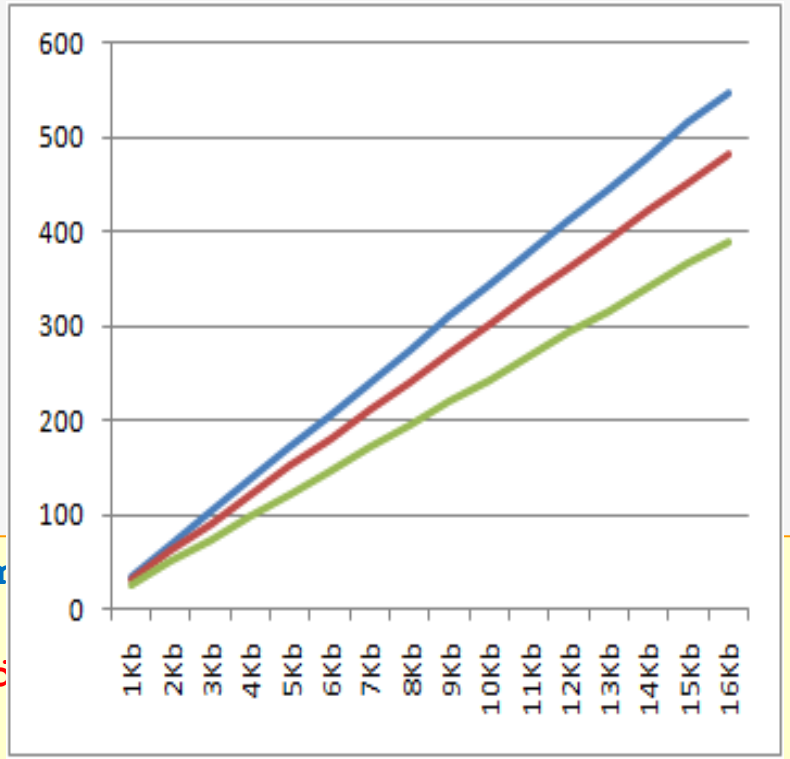
Chamar a função `length_vector` uma só vez antes do ciclo.

Menos uma chamada a função por cada iteração

Optimizar a função sum – versão 3

svector3.c

```
#include "vector.h"

void sum(Vector *v, int *dest) {
  int i, val, len;
  *dest = 0;
  len = length_vector(v);
  for (i = 0; i < len; i++) {
    elem_vector_nochk(v, i, &val);
    *dest += val;
  }
}
```



Chamar uma função de acesso ao elemento `elem_vector_nochk` que não faz verificação do índice.

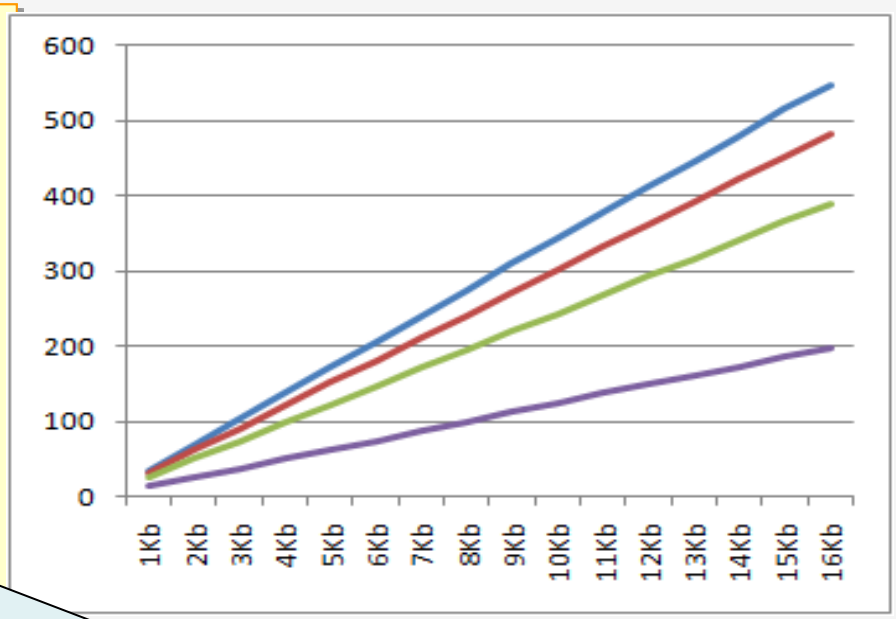
A função chamada tem melhor desempenho

```
int elem

  if (id                                    ;
  *dest
  return 1;
}

void elem_vector_nochk(Vector *v,
                       int idx, int *dest) {
  *dest = v->vec[idx];
}
```

Optimizar a função sum – versão 4

svector4.c

```
#include "vector.h"

void sum(Vector *v, int *dest) {
  int i, len, *p;
  *dest = 0;
  len = length_vector(v);
  p = start_vector(v);
  for (i = 0; i < len ; i++) {
    *dest += p[i];
  }
}
```



Chamar uma função de acesso ao *array* `start_vector` que pode ser chamada fora do *loop*.

```
int * start_vector(Vector *v) {
  return v->vec;
}
```

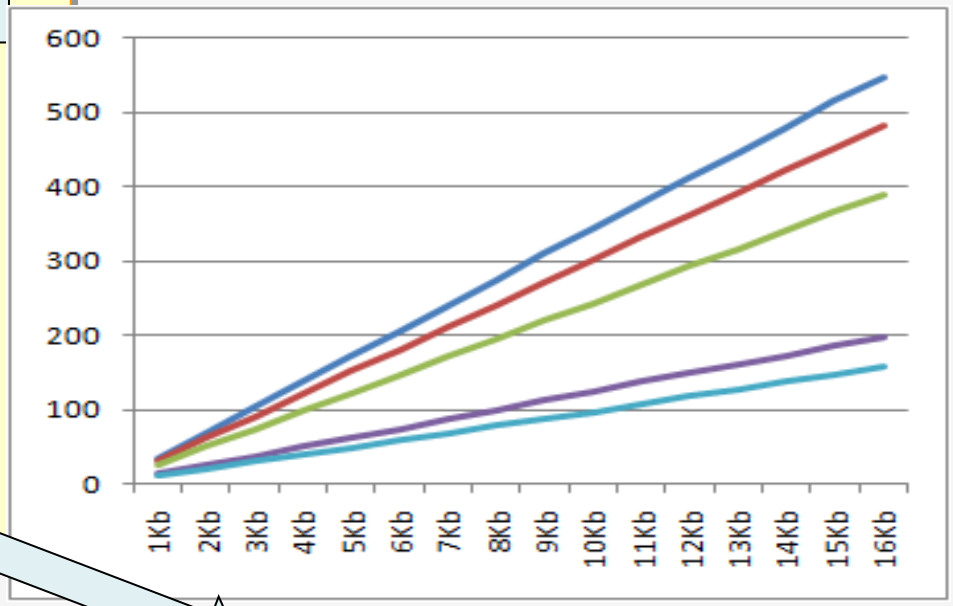
```
loop:
  mov eax, [ecx+edx*4]
  add [esi], eax
  add edx, 1
  cmp edi, edx
  jg  loop
```

Menos uma chamada a função por cada iteração

Optimizar a função sum – versão 5

svector5.c

```
#include "vector.h"

void sum(Vector *v, int *dest) {
  int i, len, *p, res;
  res = 0;
  len = length_vector(v);
  p = start_vector(v);
  for (i = 0; i < len ; i++)
    res += p[i];
  *dest = res;
}
```



Acumular o resultado numa variável local `res`.

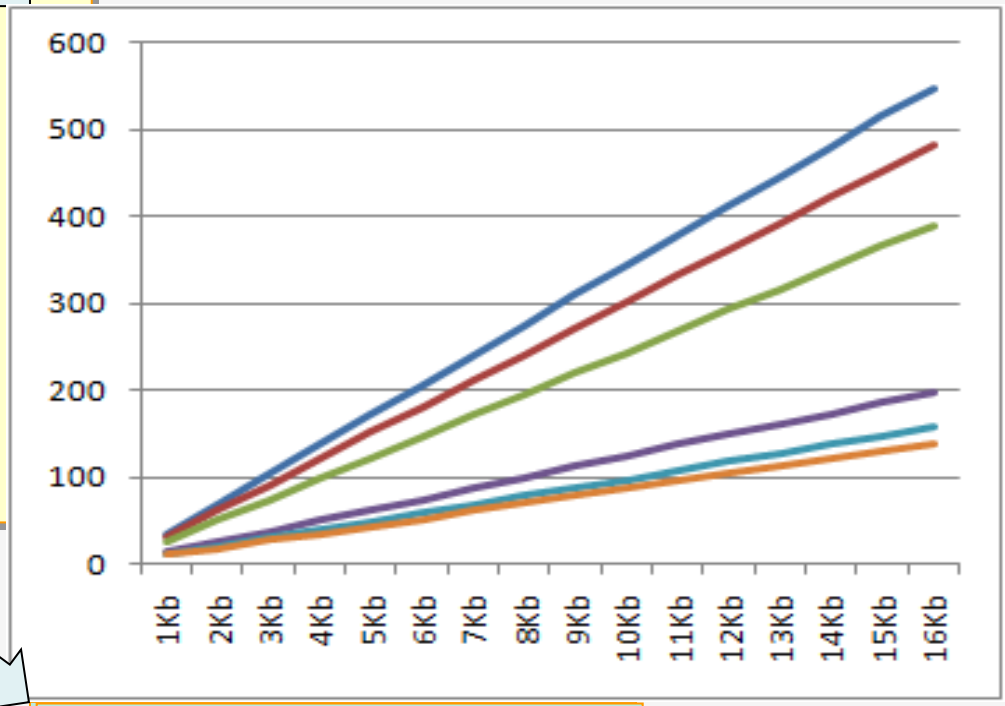
Menos um acesso para leitura e escrita em memória por cada iteração

```
loop:
  add ebx, [ecx+edx*4]
  add edx, 1
  cmp edi, edx
  jg  loop
```

Optimizar a função sum – versão 6

svector6.c

```
#include "vector.h"

void sum(Vector *v, int *dest) {
  int *p, res, *e;
  res = 0;
  p = start_vector(v);
  e = p+length_vector(v);
  for (; p < e ; p++)
    res += *p;
  *dest = res;
}
```



Usar ponteiros
em vez de índices.

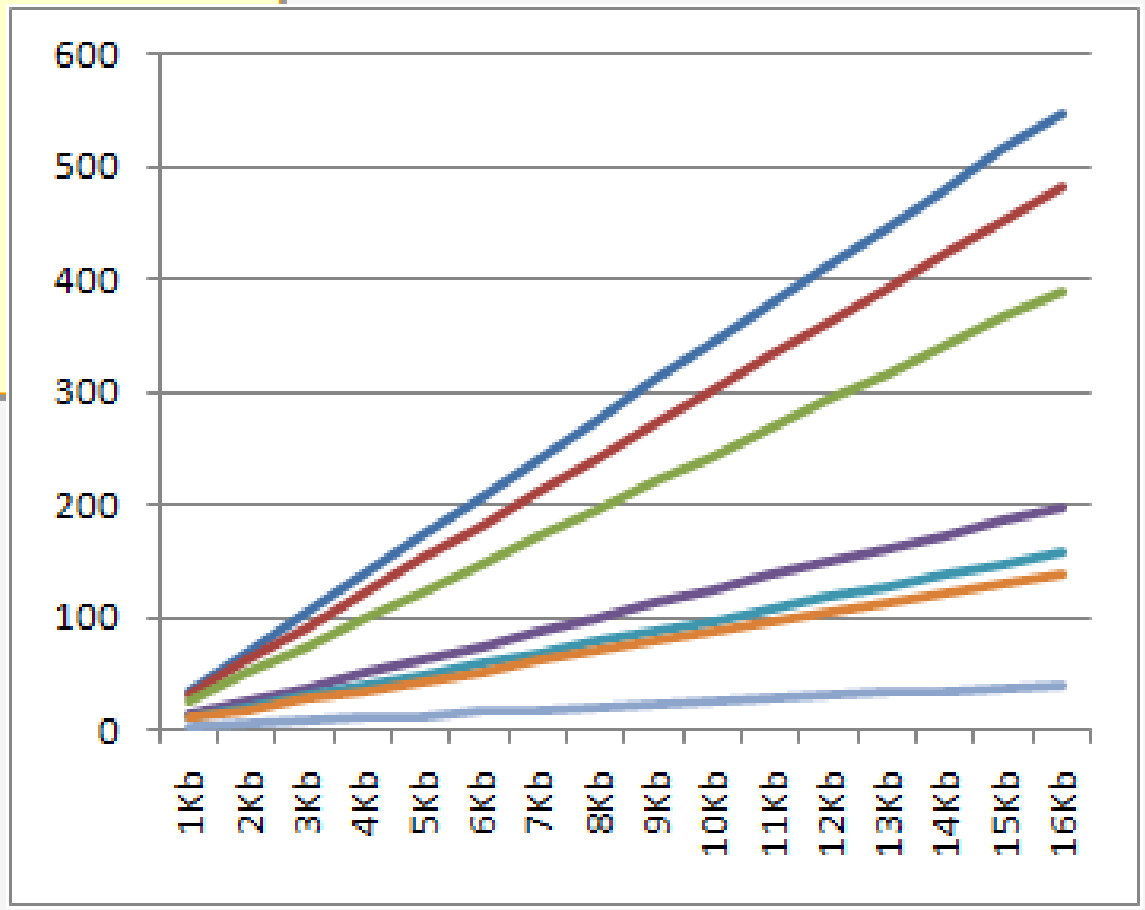
Menos aritmética por cada iteração

```
loop:
  add ebx, [esi]
  add esi, 4
  cmp edi, esi
  ja loop
```

Optimizar a função sum – versão 6 (com –O2)

svector6.c

```
#include "vector.h"

void sum(Vector *v, int *dest) {
  int *p, res, *e;
  res = 0;
  p = start_vector(v);
  e = p+length_vector(v);
  for (; p < e ; p++)
    res += *p;
  *dest = res;
}
```



Ligar a opção de optimização do compilador
-O2