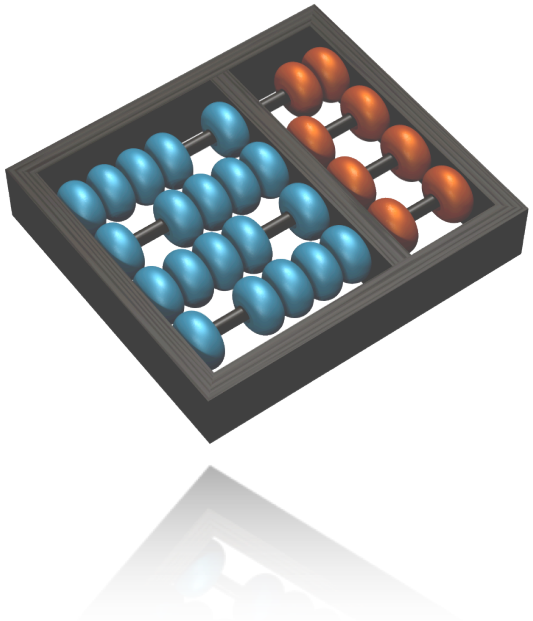
Técnicas para desenvolvimento e aceleração de códigos científicos



Prof. Edson Borin
edson@ic.unicamp.br
Instituto de Computação
UNICAMP

Minicurso
LNCC 2014

Sobre o minicurso

Segunda	Terça	Quarta	Quinta	Sexta
Introdução	Perfilamento - Contagem de tempo	Otimizações simples / compilação	Perfilamento - Detecção de código quente	GDB
Organização de processadores modernos	Otimização de acesso a dados	Bibliotecas otimizadas	SVN + CMake	Valgrind
Introdução ao laboratório		Vetorização de código		

Agenda

- Otimizações Simples
- Otimizações na Compilação
- Vetorização de Código
- Bibliotecas otimizadas

Otimizações Simples

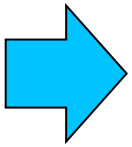
Faça menos trabalho!

```
bool flag = false;
for (i=0; i<N; i++) {
      if (A[i] > 5.0)
            flag = true;
}
return flag;
```

Otimizações Simples

Faça menos trabalho!

```
bool flag = false;
for (i=0; i<N; i++) {
    if (A[i] > 5.0)
        flag = true;
}
return flag;
```



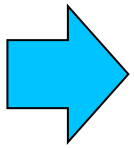
```
for (i=0; i<N; i++) {
    if (A[i] > 5.0)
        return true;
}
return false;
```

Otimizações Simples

Evite operações custosas!

```
int ang;

for (...) {
      ang = calcula_angulo();
      res += tan(ang);
}
```



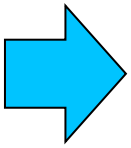
```
int ang;

for (...) {
      ang = calcula_angulo();
      res += tan_table[ang%360];
}
double tan_table[] = {
      0.0,
      ....
}
```

Otimizações Simples

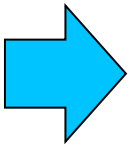
Quando possível, diminua o tamanho dos dados!

```
double a[N];
double b[N];
double c[N];
```



```
float a[N];
float b[N];
float c[N];
```

```
int IDs[N];
```



```
short IDs[N];
```

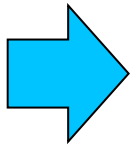
Cabem mais dados nas *caches*!

Mais operações por ciclo em instruções SIMD.

Otimizações Simples

Elimine subexpressões comuns.

```
int r, s;
...
for (i=0; i<N; i++) {
      A[i] = A[i] + r + s;
}
```

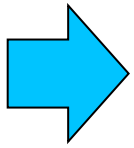


```
int r, s;
...
int tmp = r+s;
for (i=0; i<N; i++) {
      A[i] = A[i] + tmp;
}
```

Otimizações Simples

Elimine subexpressões comuns.

```
int r, s;
...
for (i=0; i<N; i++) {
      A[i] = A[i] + r + s;
}
```



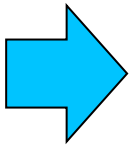
```
int r, s;
...
int tmp = r+s;
for (i=0; i<N; i++) {
      A[i] = A[i] + tmp;
}
```

Compiladores são capazes de realizar a otimização acima!

Otimizações Simples

Elimine subexpressões comuns.

```
int r, s;
...
for (i=0; i<N; i++) {
      A[i] = A[i] + r + foo();
}
```



```
int r, s;
...
int tmp = r+foo();
for (i=0; i<N; i++) {
      A[i] = A[i] + tmp;
}
```

O compilador é capaz de otimizar?

Otimizações Simples

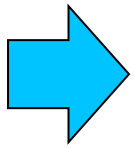
Evite saltos condicionais.

```
float sinal;
for (i=0; i<N; i++)
    for (j=0; j<N; j++) {
        if(i>j)
            sinal = 1.0;
        else if(i<j)
            sinal = -1.0;
        else
            sinal = 0.0;
        acc += sinal x A[i][j];
    }
```

Otimizações Simples

Evite saltos condicionais.

```
float sinal;
for (i=0; i<N; i++)
    for (j=0; j<N; j++) {
        if(i>j)
            sinal = 1.0;
        else if(i<j)
            sinal = -1.0;
        else
            sinal = 0.0;
        acc += sinal x A[i][j];
    }
```



```
float sinal;
for (i=0; i<N; i++)
    for (j=0; j<i; j++)
        acc += A[i][j];

for (i=0; i<N; i++)
    for (j=i+1; j<N; j++)
        acc -= A[i][j];
```

Agenda

- Otimizações Simples
- Otimizações na Compilação
- Vetorização de Código
- Bibliotecas otimizadas

Otimizações na Compilação

Use as otimizações de seu compilador.

Muitas otimizações => difícil selecionar o melhor conjunto e a melhor ordem...

Dica: comece com um conjunto padrão: **-O3**

Otimizações na Compilação

Use as otimizações de seu compilador.

Muitas otimizações => difícil selecionar o melhor conjunto e a melhor ordem...

Dica: comece com um conjunto padrão: -O3

Talvez seja necessário habilitar **mais opções**! Ex:

gcc4.7 -O3 -march=corei7-avx -mtune=corei7-avx ...

- Gera instruções **AVX**! (SIMD/256 bits)

Otimizações na Compilação

Aprenda linguagem de montagem!!!

Permite inspecionar o código gerado pelo compilador!

- gcc test.c -S -o teste.s

Otimizações na Compilação

Exemplo:

```
void foo(double *a, double *b, double s, int n) {
  for(int i=1; i<n; ++i)
    a[i] = s*b[i-1];
}
```

-O3

```
L16:
  movsd   (%rsi,%rax), %xmm3
  addl    $1, %ecx
  movhpd  8(%rsi,%rax), %xmm3
  movapd  %xmm3, %xmm1
  mulpd   %xmm2, %xmm1
  movlpd  %xmm1, 8(%rdi,%rax)
  movhpd  %xmm1, 16(%rdi,%rax)
  addq    $16, %rax
  cmpl    %ecx, %r8d
  ja      L16
```

-O3 -march=corei7-avx -mtune=corei7-avx

```
L15:
  vmovupd (%rsi,%rax), %xmm1
  addl    $1, %ecx
  vinsertf128     $0x1, 16(%rsi,%rax), ...
  vmulpd  %ymm2, %ymm1, %ymm1
  vmovupd %xmm1, 8(%rdi,%rax)
  vextractf128    $0x1, %ymm1, 24(%rdi,%rax)
  addq    $32, %rax
  cmpl    %ecx, %r8d
  ja      L15
```

Otimizações na Compilação

Exemplo:

```
void foo(double *a
  for(int i=1; i<n
    a[i] = s*b[i-1
}
```

movlpd: Move Low Packed Double-Precision Floating-Point Value

-O3

```
L16:
  movsd    (%rsi,%rax), %xmm3
  addl     $1, %ecx
  movhpd   8(%rsi,%rax), %xmm3
  movapd   %xmm3, %xmm1
  mulpd    %xmm2, %xmm1
  movlpd   %xmm1, 8(%rdi,%rax)
  movhpd   %xmm1, 16(%rdi,%rax)
  addq     $16, %rax
  cmpl     %ecx, %r8d
  ja       L16
```

-avx

```
  vmovupd (%rsi,%rax), %xmm1
  addl     $1, %ecx
  vinsertf128     $0x1, 16(%rsi,%rax), ...
  vmulpd   %ymm2, %ymm1, %ymm1
  vmovupd %xmm1, 8(%rdi,%rax)
  vextractf128    $0x1, %ymm1, 24(%rdi,%rax)
  addq     $32, %rax
  cmpl     %ecx, %r8d
  ja       L15
```

Otimizações na Compilação

Use os relatórios (*logs*) de compilação!

- gcc4.7 -O3 **-ftree-vectorizer-verbose**=1 test.c -c

Otimizações na Compilação

Use os relatórios (*logs*) de compilação!

- gcc4.7 -O3 -ftree-vectorizer-verbose=1 test.c -c

```
Analyzing loop at test.c:9
Vectorizing loop at test.c:9
9: created 2 versioning for alias checks.
9: LOOP VECTORIZED.
test.c:6: note: vectorized 1 loops in function.

Analyzing loop at test.c:16
Vectorizing loop at test.c:16
16: created 1 versioning for alias checks.
16: LOOP VECTORIZED.
test.c:13: note: vectorized 1 loops in function.
```

Agenda

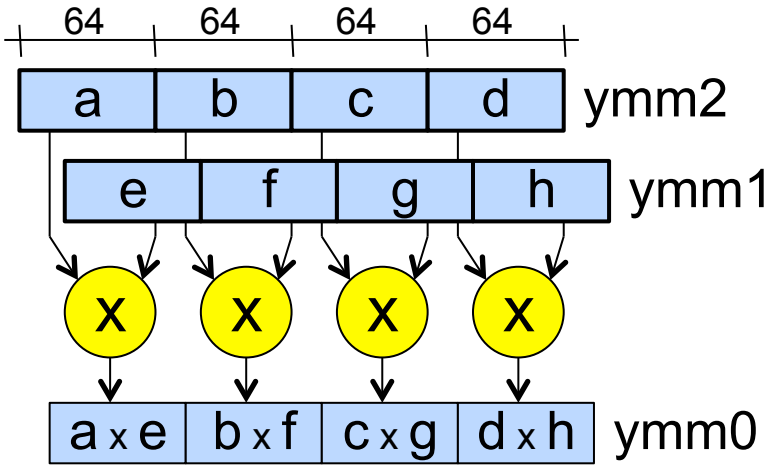
- Otimizações Simples
- Otimizações na Compilação
- Vetorização de Código
- Bibliotecas otimizadas

Vetorização de Código

Vetorização: usar instruções SIMD para realizar operações de forma paralela!

Exemplo:

```
vmulpd %ymm2,%ymm1,%ymm0
```

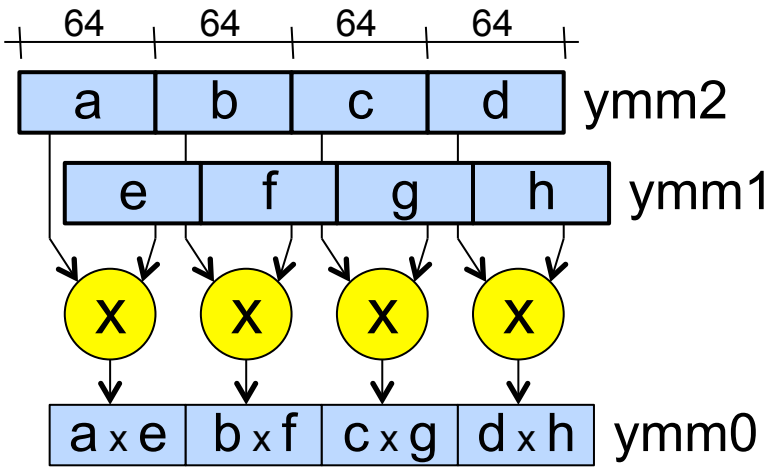


Vetorização de Código

Vetorização: usar instruções SIMD para realizar operações de forma paralela!

Exemplo:

```
vmulpd %ymm2,%ymm1,%ymm0
```



Diversos conjuntos de instruções: SSE, SSE2, ..., AVX, AVX2, NEON, ...

Vetorização de Código

Vetorização: geração de código manual: "*compiler intrinsics*"

Exemplo:

```
float produto_interno(float* a, float* b)
{
   int i;
   float total=0.0;
   for (i=0; i<SIZE; i++)
      total += a[i] * b[i];
   return total;
}
```

Vetorização de Código

Vetorização: geração de código manual: "*compiler intrinsics*"

```
for (i=0; i<SIZE; i++)
   total += a[i] * b[i];
```

```
for (i=0; i<SIZE; i+=4){
   total[0] += a[i+0] * b[i+0];
   total[1] += a[i+1] * b[i+1];
   total[2] += a[i+2] * b[i+2];
   total[3] += a[i+3] * b[i+3];
}
```

Vetorização de Código

Vetorização: geração de código manual: "*compiler intrinsics*"

```
for (i=0; i<SIZE; i+=4){
   total[0] += a[i+0] * b[i+0];
   total[1] += a[i+1] * b[i+1];
   total[2] += a[i+2] * b[i+2];
   total[3] += a[i+3] * b[i+3];
}
```

```
for (i=0; i<SIZE; i+=4){
   v1  = _mm_loadu_ps(a+i);
   v2  = _mm_loadu_ps(b+i);
   v3  = _mm_mul_ps(v1, v2);
   acc = _mm_add_ps(acc, v3);
}
```

Vetorização de Código

Vetorização: geração de código manual: "*compiler intrinsics*"

```
__m128 v1, v2, v3, acc;
acc = _mm_setzero_ps();  // acc = |0|0|0|0|
for (i=0; i<SIZE; i+=4){
   v1  = _mm_loadu_ps(a+i);
   v2  = _mm_loadu_ps(b+i);
   v3  = _mm_mul_ps(v1, v2);
   acc = _mm_add_ps(acc, v3);
}
```

Vetorização de Código

Vetorização: geração de código manual: "*compiler intrinsics*"

```
__m128 v1, v2, v3, acc;
acc = _mm_setzero_ps();  // acc = |0|0|0|0|
for (i=0; i<SIZE; i+=4){
   v1  = _mm_loadu_ps(a+i);
   v2  = _mm_loadu_ps(b+i);
   v3  = _mm_mul_ps(v1, v2);
   acc = _mm_add_ps(acc, v3);
}
acc = _mm_hadd_ps(acc,acc);
acc = _mm_hadd_ps(acc,acc);
_mm_store_ss(&total,acc);
return total;
```

Vetorização de Código

Vetorização: geração de código manual: "*compiler intrinsics*"

```
#include <x86intrin.h>
void produto_interno(float* a, float* b, int n) {
   int i; float total;
   __m128 v1, v2, v3, acc;
   acc = _mm_setzero_ps();  // acc = |0|0|0|0|
   for (i=0; i<SIZE; i+=4){
      v1  = _mm_loadu_ps(a+i);
      v2  = _mm_loadu_ps(b+i);
      v3  = _mm_mul_ps(v1, v2);
      acc = _mm_add_ps(acc, v3);
   }
   acc = _mm_hadd_ps(acc,acc);
   acc = _mm_hadd_ps(acc,acc);
   _mm_store_ss(&total,acc);
   return total;
}
```

Vetorização de Código

Vetorização: geração de código automática pelo compilador!

- A maioria dos compiladores modernos já faz automaticamente.

- gcc4.7 -O3 -ftree-vectorizer-verbose=1 test.c –c

Vetorização de Código

Vetorização: geração de código automática pelo compilador!

- A maioria dos compiladores modernos já faz automaticamente.

- gcc4.7 -O3 -ftree-vectorizer-verbose=1 test.c –c

Dicas:

- minimize o uso de expressões condicionais dentro do laço.
- simplifique o controle do laço

Vetorização de Código

Vetorização: geração de código automática pelo compilador!

Exemplo:

```
void soma_vetor(float* a, float* end_a,
                float* b, float* end_b)
{
  while( (a < end_a) && (b < end_b) )
    *(a++) += *b++;
}
```

Vetorização de Código

Vetorização: geração de código automática pelo compilador!

Exemplo:

```
void soma_vetor(float* a, float* end_a,
```

```
gcc -O3 -ftree-vectorize -ftree-vectorizer-
verbose=5 -Wall soma_vetor_v1.c -c

Analyzing loop at soma_vetor_v1.c:2
soma_vetor_v1.c:1: note: vectorized 0 loops
in function.
```

Vetorização de Código

Vetorização: geração de código automática pelo compilador!

```
while( (a < end_a) && (b < end_b) ) {
    *(a++) += *b++;
}
```

```
int n = MIN(end_a-a, end_b-b);
int i;
for (i=0; i<n; i++) {
   *(a++) += *b++;
}
```

Vetorização de Código

Vetorização: geração de código automática pelo

```
gcc -O3 -ftree-vectorize -ftree-vectorizer-verbose=1 -
Wall soma_vetor_v2.c -c
Analyzing loop at soma_vetor_v2.c:6
Vectorizing loop at soma_vetor_v2.c:6
6: created 1 versioning for alias checks.
6: LOOP VECTORIZED.
soma_vetor_v2.c:3: note: vectorized 1 loops in function.
```

```
int n = MIN(end_a-a, end_b-b);
int i;
for (i=0; i<n; i++) {
   *(a++) += *b++;
}
```

Agenda

- Otimizações Simples
- Otimizações na Compilação
- Vetorização de Código
- Bibliotecas otimizadas

Bibliotecas Otimizadas

Em muitos casos, é muito difícil gerar código ótimo (ou bom) para processadores modernos!

- Compilador não consegue inferir informações!
- Ausência de informação da microarquitetura!
- etc...

Gerar código em ling. de montagem é um desafio!

Solução: usar bibliotecas otimizadas!

Bibliotecas Otimizadas

Exemplo: BLAS – *Basic Linear Algebra Subprograms*

operações entre vetores e matrizes!

BLAS nível 1: operações vetor/vetor

BLAS nível 2: operações vetor/matriz

BLAS nível 3: operações matriz/matriz

- Exemplo: DGEMM (DP - General Matrix Multiply)

Bibliotecas Otimizadas

Exemplo: BLAS – *Basic Linear Algebra Subprograms*

Muito utilizada em computação de alto desempenho

Geralmente fornecida pelo fabricante do *Hardware*

- *Intel MKL, AMD ACML, ...*

Atividades de laboratório

www.ic.unicamp.br/~edson/disciplinas/lncc14/index.html

- Bibliotecas otimizadas
- Vetorização de código
- Otimizações do compilador