4th Internal Conference on Computer Architecture (ICCA'03)

Speed-up Techniques in Matrix Computation: a Case Study

MI/CEI
2002/2003

4th Internal Conference on Computer Architecture (ICCA'03)

Introdução

Metodologias:

- **Ferramentas**
- **Algoritmos**
- **Metodologia de Avaliação**

Dados Recolhidos

Análise dos Dados

Conclusões

4th Internal Conference on Computer Architecture (ICCA'03)

Introdução:

- **Falta de qualidade do software**
- **Qualidade subestimada para "ganhar" velocidade na produção de software**
- **Analise de diferentes técnicas de programação e avaliar as diferenças**

4th Internal Conference on Computer Architecture (ICCA'03)

Introdução (cont.):

- **Análise com base na arquitectura do processador e memória**
  - **Multiplicação de duas matrizes quadradas**
  - **5 algoritmos diferentes**
  - **Diferentes níveis de optimização**

Metodologias - Ferramentas:

- **Linguagem de Programação:**
  - **C**
- **Compilador**
  - **GCC da GNU**
- **Bibliotecas do BLAS**
  (Basic Linear Algebra Subprograms)

Metodologias – Ferramentas (Cont.):

- **Medições em CPE (Cycles per Elements)**
  - **Métrica mais usada**
  - **Permite comparar matrizes com diferentes dimensões**
  - **Métrica indicada para medição de tempos entre dois pontos do código**
- **Algoritmos baseados em "loops"**
- **Método de medição praticamente independente do processador**

Metodologias – Ferramentas (Cont.):

- **Cálculo feito entre a chamada do `start_counter` e a chamada do `get_counter`**
- **É devolvido o número de ciclos decorridos**

Metodologias – Algoritmos:

- **5 Algoritmos**
  - **Normal**
    Normal
  - **Transpose**
    Transpose
  - **Unroll16**
    Unroll16
  - **Block (com sub-blocos de dimensão 20)**
    Block
  - **BLAS**
    `cblas_dgemm`

```
void normal(int N){
      int i;

      for (i=0;i<N;i++){
                for (j=0;j<N;j++){
                          c[i][j] = 0.0;
                          for (k=0;k<N;k++){
                                    c[i][j] += a[i][k]*b[k][j];
                          }
                }
      }
}
```



```
void transpose(int N){
         int i,j,k;
         double temp;

                   for (i=0;i<N;i++){
                   for (j=0;j<N;j++){
                             bt[j][i] = b[i][j];
                   }
         }

         for (i=0;i<N;i++){
                   for (j=0;j<N;j++){
                             temp = a[i][0]*bt[j][0];
                             for (k=1;k<N;k++){
                                       temp += a[i][k]*bt[j][k];
                             }
                             c[i][j] = temp;
                   }
         }
   }
```



```
void unroll16(int N){
int i,j,k;
double temp;
for (i=0;i<N;i++){
    for (j=0;j<N;j++){
          temp = 0.0;
          for (k=0;k<(N-15);k+=16){
                    temp += a[i][k]*b[k][j];
                              temp += a[i][k+1]*b[k+1][j];
                              temp += a[i][k+2]*b[k+2][j];
                              temp += a[i][k+3]*b[k+3][j];
                              ……..
                              temp += a[i][k+15]*b[k+15][j];
          }
          for (;k<N;k++){
                    temp += a[i][k]*b[k][j];
          }
          c[i][j] = temp;
    }
}
}
```



```
void blocking(int N, int nb) {

for( ii = 0; ii < N; ii += nb ){
    for( jj = 0; jj < N; jj += nb ){
         for( i = ii; i < ii + nb ; i++ ){
              for( j = jj; j < jj + nb ; j++ ){
                   c[i][j] = 0.0;
                   }
         for( kk = 0; kk < N; kk += nb ){
           for( i = ii; i < ii + nb ; i += 2 ){
                for( j = jj; j < jj + nb ; j += 2 ){
                              s00 = c[i ][j ];
                              s01 = c[i ][j+1];
                              s10 = c[i+1][j ];
                              s11 = c[i+1][j+1];
                             for( k = kk; k < kk + nb ; k++ ){
                                  s00 = s00 + a[i ][k]*b[k][j ];
                                  s01 = s01 + a[i ][k]*b[k][j+1];
                                  s10 = s10 + a[i+1][k]*b[k][j ];
                                  s11 = s11 + a[i+1][k]*b[k][j+1];
                   }
                   c[i ][j ] = s00;
                              c[i ][j+1] = s01;
                              c[i+1][j ] = s10;
                              c[i+1][j+1] = s11;
```

4th Internal Conference on Computer Architecture (ICCA'03)

Metodologias – Metodologia de Avaliação:

- **Linux**
- **Processador 1133 MHz Celeron®**
- **Memória Principal:**
  - **256 MB**
- **Caches**
  - **L1:**
    - **32 Kb**
  - **L2:**
    - **256 Kb**

4th Internal Conference on Computer Architecture (ICCA'03)

Metodologias – Metodologia de Avaliação (Cont.):

- **Matrizes de dimensão:**
  - **20**
    - **Cabe na cache L1**
    - **Tem 400 elementos com 8 bytes cada = 6.4Kb total**
  - **100**
    - **Cabe em L2 mas não cabe em L1**
    - **Tem 10000 elementos com 8 bytes cada = 160Kb total**
  - **200**
    - **Não cabe em L2 nem cabe em L1**
    - **Tem 40000 elementos com 8 bytes cada = 640Kb total**

4th Internal Conference on Computer Architecture (ICCA'03)

Metodologias – Metodologia de Avaliação (Cont.):

- **Níveis de optimização:**
  - **O0**
    - **Não faz qualquer optimização no código**
  - **O2**
    - **executa todas as optimizações suportadas excepto no loop unrolling e na função inlinig**
- **Medições em CPE**
  - **Exemplo:**
    - **Matrizes de dimensão NxN precisamos:**
      - **NxN valores da matriz resultado**
      - **Nx2 elementos da matriz fonte por cada elemento**
      - **Total: NxNxNx2**

4th Internal Conference on Computer Architecture (ICCA'03)

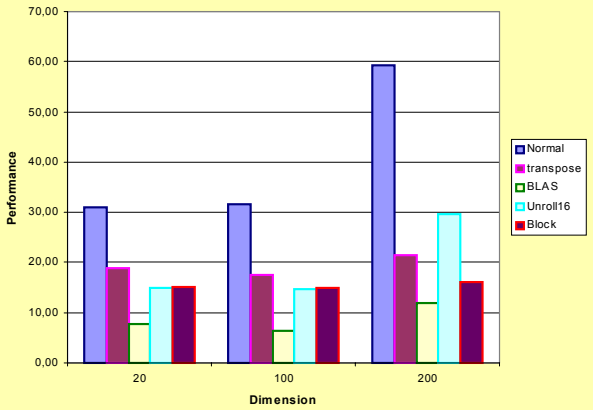
Dados Recolhidos :

- **Cálculo da média de cinco resultados obtidos por cada algoritmo e para cada uma das dimensões**
- **Análise sobre dois pontos de vista:**
  - **Comparação da performance entre os vários algoritmos**
  - **Análise das melhorias de performance provocadas pelos diferentes níveis de optimização do compilador**

Dados Recolhidos (Cont.) :

- **Resultados obtidos com GCC O0:**
  - **BLAS mais eficiente para as 3 dimensões**
  - **Block com eficiência perto da do BLAS**
  - **Normal o menos eficiente**
    - **Normal com 59,35 CPE e o BLAS com 11,86 CPE (matrizes de dimensão 200)**

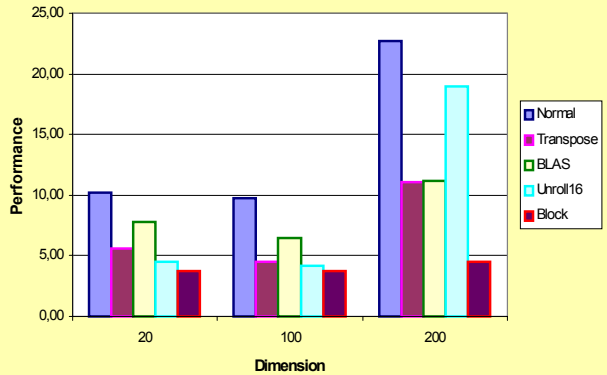
Dados Recolhidos (Cont.) :



Dados Recolhidos (Cont.) :

- **Resultados obtidos com GCC O2:**
  - **Block mais eficiente**
    - **4,45 CPE para matrizes de dimensão 200**
    - **Valor do CPE desceu para menos de metade do necessário pelo BLAS na compilação com GCC O0**
  - **Unroll e o Transpose:**
    - **Quase a mesma eficiência que o Block mas apenas para matrizes de dimensão 20 e 100**
  - **Normal teve uma melhoria de performance mas continua a ser o menos eficiente.**

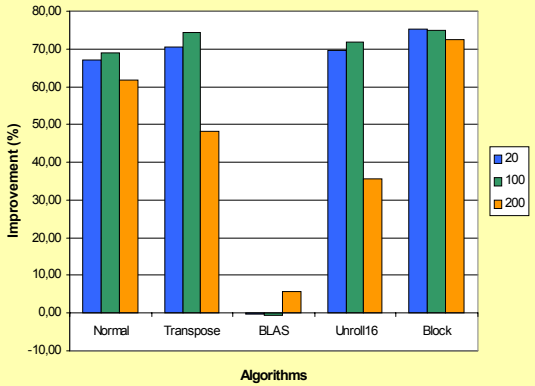
Dados Recolhidos (Cont.) :



Dados Recolhidos (Cont.) :

- Da análise dos resultados concluímos:
  - O BLAS é o melhor algoritmo quando compilado com o GCC O0
  - Quando compilado com o GCC O2 piora os resultados quando comparado com os outros algoritmos
  - Conclusão: Porque como estamos a usar uma função externa não é sujeito ás optimizações do compilador

Dados Recolhidos (Cont.) :



Dados Recolhidos (Cont.) :

- Comparação entre GCC O0 e GCC O2:
  - Melhorias em todos os algoritmos excepto no BLAS:
    - O Normal teve uma melhoria de:
      - ≈ 67% nas matrizes de dimensão 20
      - ≈ 69% nas matrizes de dimensão 100
      - ≈ 62% nas matrizes de dimensão 200
    - O Block:
      - Melhoria de ≈ 75% nas matrizes de dimensão 20
      - Deterioração de ≈ 73% nas matrizes de dimensão 200

Dados Recolhidos (Cont.) :

- Resultados da multiplicação das matrizes de dimensão 100 é idêntico ou melhor que os resultados da multiplicação das matrizes de dimensão 20
- Dado curioso:
  - a matriz de dimensão 20 cabe na cache L1 e a de dimensão 100 cabe na cache L2
- Justificação:
  - Para este processador específico a cache de nível L2 está no processador e é acedida à mesma velocidade do processador

4th Internal Conference on Computer Architecture (ICCA'03)

Análise dos Dados:

- **Ciclos são a principal fonte de ineficiência**
- **Principais factores que influenciam o desempenho:**
  - **Técnicas de Escrita**
  - **Comportamento da memória**
- **Análise dos Algoritmos:**
  - **Normal tem um elevado *miss rate***
  - **Para cada elemento de B tenho *cache miss***

4th Internal Conference on Computer Architecture (ICCA'03)

Análise dos Dados (Cont.):

- **Análise dos Algoritmos:**
  - **Transpose tem um pequeno *miss rate***
  - **De cada vez que ocorre um cache miss tenho 3 cache hits**
  - **A matriz B é percorrida ao longo da linha.**

4th Internal Conference on Computer Architecture (ICCA'03)

Análise dos Dados (Cont.):

- **Análise dos Algoritmos:**
  - **Unroll16 é mais eficaz com adições (por causa da latência das operações)**
  - **A latência tem origem nas limitações do paralelismo**
  - **Eliminar três operações por cada ciclo *for***

4th Internal Conference on Computer Architecture (ICCA'03)

Análise dos Dados (Cont.):

```
…
for (k=0;k<(N-15);k+=16){
            temp += a[i][k]*b[k][j];
            temp += a[i][k+1]*b[k+1][j];
            temp += a[i][k+2]*b[k+2][j]
…
```

```
...
movl    20(%esp), %eax
leal    0(,%eax,4), %ecx
movl    a, %edx
movl    12(%esp), %eax
leal    0(,%eax,8), %esi
movl    (%edx,%ecx), %edi
movl    12(%esp), %eax
leal    0(,%eax,4), %ebx
movl    b, %ecx
movl    16(%esp), %eax
leal    0(,%eax,8), %edx
movl    (%ecx,%ebx), %eax
fldl    (%edi,%esi)
fmull   (%eax,%edx)
fldl    (%esp)
faddp   %st, %st(1)
fstpl   (%esp)
...
```

Análise dos Dados (Cont.):

- **Análise dos Algoritmos:**
  - **Block é calculado recorrendo a sub blocos de matrizes**
  - **Dimensão dos sub blocos n=20 (cabe em cache L1)**
  - **Acessos à memória muito mais rápidos pois é como as submatrizes estejam todas em cache L1**

Análise dos Dados (Cont.):

- **As optimizações de um algoritmo não implicam a criação de um novo algoritmo ou das optimizações do compilador**
- **Pequenos ajustes podem melhorar o desempenho**
- **Para testarmos esses ajustes fizeram-se alterações no algoritmo normal**

Análise dos Dados (Cont.):

- **Alterou-se o algoritmo normal para o seguinte:**

```
void normal_1(int N) {
  int i,j,k;
  double temp;

  for (i=0;i<N;i++){
        for (j=0;j<N;j++){
              c[i][j] = 0.0;
              for (k=0;k<N;k++){
                    temp += a[i][k]*b[k][j];
              }
              c[i][j]=temp;
        }
  }
}
```


Análise dos Dados (Cont.):

- **Vantagem:**
  - **A variável temporária é guardada em registo**
  - **Maior rapidez no acesso á variável**
  - **Evita muitos acessos à memória**

Análise dos Dados (Cont.):

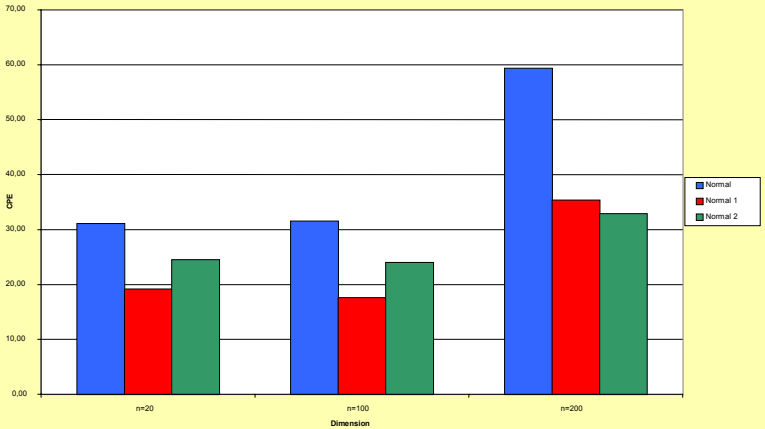
- **Segunda alteração do algoritmo normal**

```
void normal2(int N) {
  int i,j,k;
  double r;
        for (k=0;k<N;k++){
                for (i=0;i<N;i++){
                        r = a[i][k];
                        for (j=0;j<N;j++){
                                c[i][j] += r*b[k][j];
                        }
                }
        }
}
```


Análise dos Dados (Cont.):

- **Vantagem:**
  - **Baixo miss rate porque matriz B é acedida por linha.**
  - **Maior rapidez no acesso á variável**
  - **Evita muitos acessos à memória**
- **Desvantagens :**
  - **É necessário mais um acesso à memoria para escrita**

Análise dos Dados (Cont.):



Conclusões:

- **A optimização do desempenho depende de vários factores:**
  - **Do hardware**
  - **Do Código desenvolvido**
  - **Do compilador e dos níveis de optimização seleccionados**
- **Factores que não actuam por si, mas todos eles em conjunto**
- **Fazendo alterações em cada um deles pode levar á melhoria do desempenho**