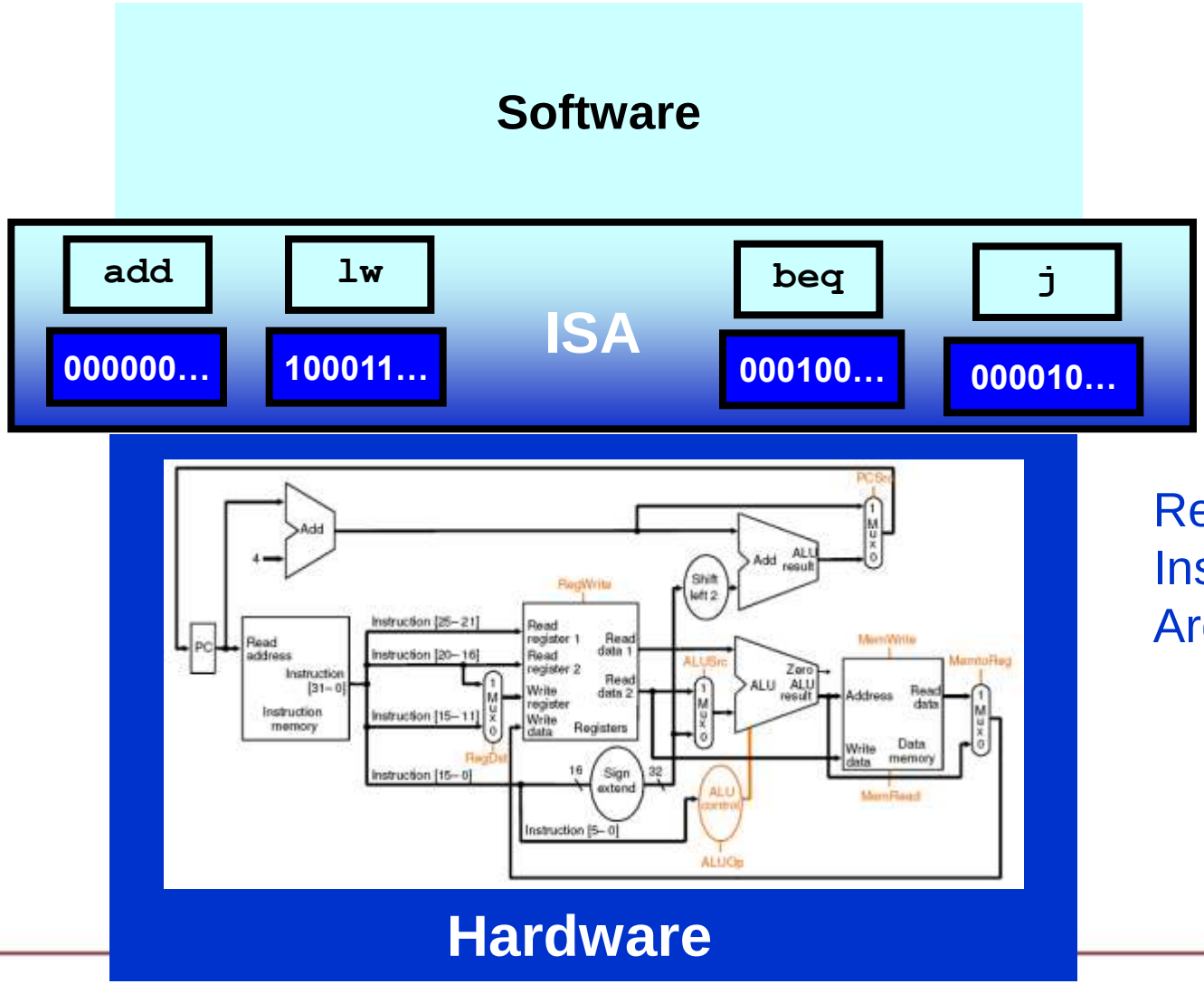
Infraestrutura de Hardware

Desempenho

Perguntas que Devem ser Respondidas ao Final do Curso

- Como um programa escrito em uma linguagem de alto nível é entendido e executado pelo HW?
- Qual é a interface entre SW e HW e como o SW instrui o HW a executar o que foi planejado?
- **O que determina o desempenho de um programa e como ele pode ser melhorado?**
- Que técnicas um projetista de HW pode utilizar para melhorar o desempenho?

Interface HW/SW: ISA



Repertório de Instruções da Arquitetura

Analisando Desempenho

- Desempenho é um importante fator de qualidade de um sistema
- Análise difícil
  - Tamanho e complexidade de softwares atuais
  - Técnicas sofisticadas utilizadas no projeto do hardware para aumento de desempenho
- Diferentes métricas podem ser utilizadas dependendo do tipo de aplicação
- Diferentes aspectos do sistema podem ter impacto maior no desempenho
  - Entendimento de como os diferentes aspectos afetam o sistema é essencial para desenvolver software ou projetar hardware com melhor desempenho

Fatores Que Influenciam Desempenho

- **Algoritmo**

  Determina número de operações executadas

- **Linguagem de Programação, compilador, arquitetura**

  Determina número de instruções de máquina executadas por operação

- **Processador e estrutura de memória**

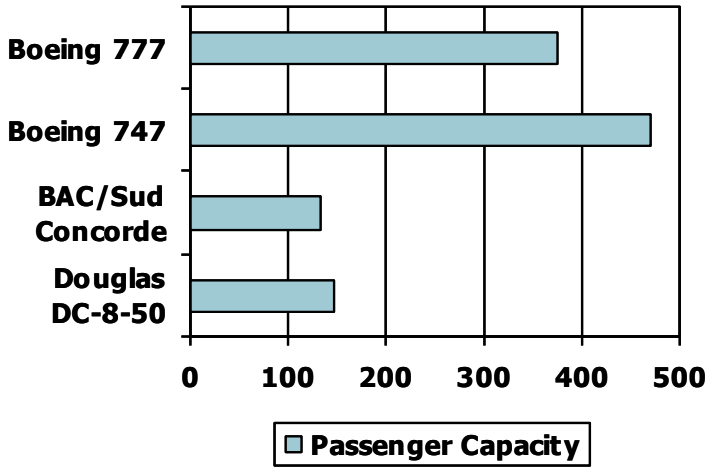
  Determina velocidade em que instruções são executadas

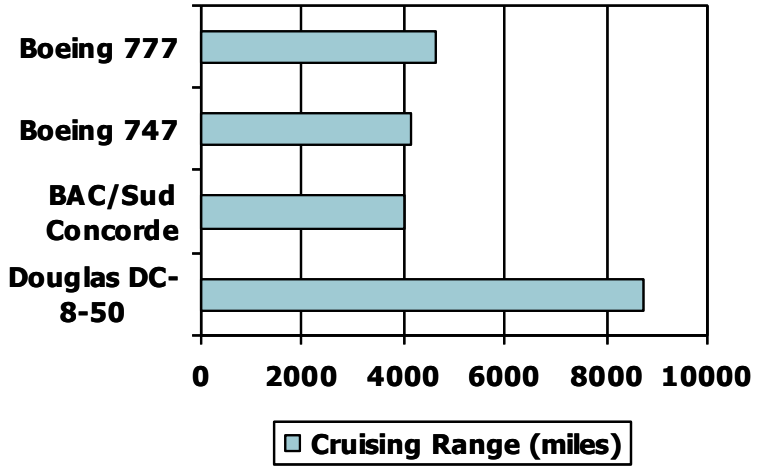
- **E/S (incluindo Sistema Operacional)**

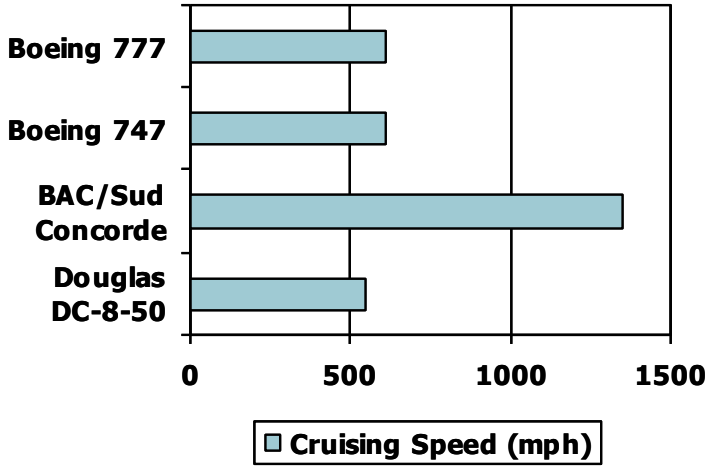
  Determina velocidade em que operações de E/S são executadas

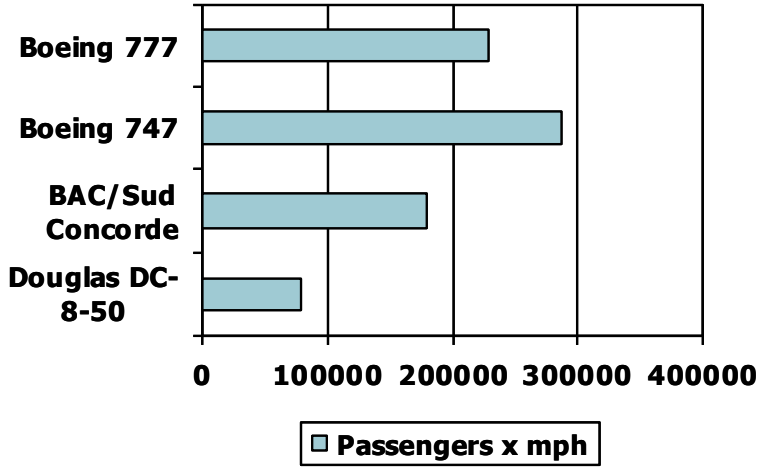
Definindo Desempenho

Qual avião tem melhor desempenho?









Diferentes Métricas

- **Tempo de Execução ou Resposta**
  - Tempo que leva para completar uma tarefa
  - Importante para o usuário comum de um sistema
- **Throughput**
  - Trabalho total feito por unidade de tempo
    - Exemplos: tarefas ou transações por segundo ou hora
  - Importante para análise de máquinas servidoras
  - Frequentemente afetado pelo tempo de execução

Analisaremos fatores que afetam tempo de execução

Desempenho Relativo

- **Desempenho = 1/Tempo de Execução**
- **"X é *n* vezes mais rápido que Y"**

$\text{Desempenho}_x / \text{Desempenho}_y = \text{Tempo de Execução}_y / \text{Tempo de Execução}_x = n$

- **Exemplo: tempo para rodar um programa**

  10s em A, 15s em B

  $\text{Tempo de Execução}_B$ / $\text{Tempo de Execução}_A$ = 15s / 10s = 1.5

  Então A é 1.5 vezes mais rápido que B

Medindo Tempo de Execução

- **Tempo Gasto (Elapsed time)**

  Tempo total de resposta, incluindo todos os aspectos

  - Processamento, E/S, overhead de sistema operacional, tempo esperando algum evento

  Determina desempenho do sistema

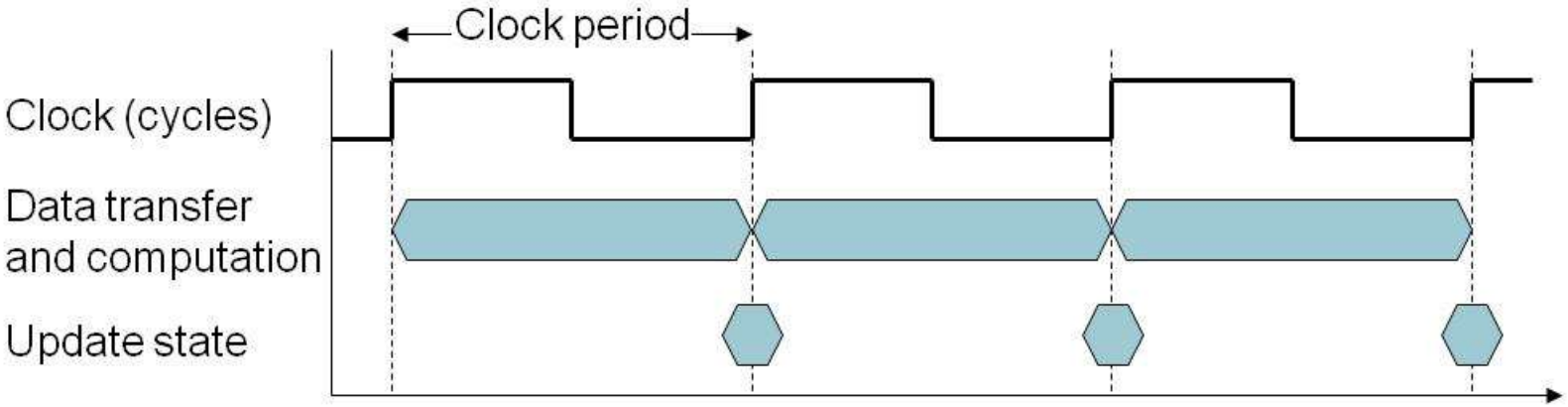
- **Tempo de CPU (CPU Time)**

  Tempo de processamento de um programa

  - Não leva em conta tempo de E/S, execução de outros programas, etc

Clock

- Operações de uma CPU são governadas pelo clock



- Período do clock: duração de um ciclo de clock

  Ex: 250ps = 0.25ns = 250×10^{-12}s

- Frequência do Clock : ciclos por segundo

  Ex: 4.0GHz = 4000MHz = 4.0×10^{9}Hz

Tempo de CPU

Tempo de CPU = Ciclos de Clock (utilizados pela CPU) x Período de Clock

= Ciclos de Clock / Frequência

- Desempenho pode ser melhorado
  - Reduzindo número de ciclos de clock
  - Aumentando frequência do clock
- Projetista de Hardware deve frequentemente fazer o *trade off* entre frequência e número de ciclos

Tempo de CPU: Exemplo

- Computador A

  Frequência de clock: 2GHz, Tempo de CPU: 10s

- Projetando Computador B

  Objetivo: Tempo de CPU = 6s

  Maior frequência do clock possível, mas isto causa um aumento no número de ciclos em 1,2x

- **Qual deve ser então a frequência do clock para atingir o tempo de CPU necessário?**

$$\text{Frequência}_B = \frac{\text{Ciclos de Clock}_B}{\text{Tempo de CPU}_B} = \frac{1{,}2 \times \text{Ciclos de Clock}_A}{6\text{s}}$$

$$\text{Ciclos de Clock}_A = \text{Tempo de CPU}_A \times \text{Frequência}_A$$
$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Frequência}_B = \frac{1{,}2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

Número de Instruções e CPI

Ciclos de Clock = Nº de Instruções x Ciclos por Instrução (CPI)

Tempo de CPU = Nº de Instruções x CPI x Período de Clock

$$= \frac{\text{Nº de Instruções x CPI}}{\text{Frequência}}$$

- **Nº de Instruções de um programa**
  - Determinado pelo algoritmo, ISA e compilador
- **Quantidade média de ciclos por instrução**
  - Determinado pelo hardware da CPU
  - Se diferentes instruções têm diferentes CPIs
    - Depende de como as diferentes instruções foram usadas no programa

CPI : Exemplo

- Computador A: Tempo do ciclo = 250ps, CPI = 2,0
- Computador B: Tempo do ciclo = 500ps, CPI = 1,2
- Mesma ISA
- **Qual é o mais rápido e quão mais rápido?**

Tempo de CPU_A = Nº de Instruções x CPI_A x Tempo do $Ciclo_A$
= N x 2,0 x 250ps = N x 500ps ← A é mais rápido

Tempo de CPU_B = Nº de Instruções x CPI_B x Tempo do $Ciclo_B$
= N x 1,2 x 500ps = N x 600ps

$$\frac{\text{Tempo de CPU}_B}{\text{Tempo de CPU}_A} = \frac{\text{N x 600ps}}{\text{N x 500ps}} = 1{,}2$$ ← 1,2x mais rápido

Mais Detalhes sobre CPI

- Se diferentes classes de instruções levam diferentes números de ciclos

$$\text{Ciclos de Clock} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Nº de instruções}_i)$$

- Média ponderada da CPI

$$\text{CPI} = \frac{\text{Ciclos de Clock}}{\text{Nº de Instruções}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{Nº de Instruções}_i}{\text{Nº de Instruções}} \right)$$

Mais um Exemplo de CPI

- Uma mesma sequência de código foi compilado de 2 formas diferentes usando instruções das classes A, B, C

Classe	A	B	C
CPI da classe	1	2	3
Nº de Instruções na sequência 1	2	1	2
Nº de Instruções na sequência 2	4	1	1

- **Nº de Instruções na sequência 1 = 5**

  Ciclos de Clock
  $= 2 \times 1 + 1 \times 2 + 2 \times 3$
  $= 10$

  Média CPI = 10/5 = 2,0

- **Nº de Instruções na sequência 2 = 6**

  Ciclos de Clock
  $= 4 \times 1 + 1 \times 2 + 1 \times 3$
  $= 9$

  Média CPI = 9/6 = 1,5

Desempenho: Resumindo

$$\text{Tempo de CPU} = \frac{\text{Instruções}}{\text{Programa}} \times \frac{\text{Ciclos de Clock}}{\text{Instrução}} \times \frac{\text{Segundos}}{\text{Ciclo de Clock}}$$

- **Desempenho depende de:**
  - Algoritmo: afeta nº de instruções, possivelmente CPI
  - Linguagem: afeta nº de instruções, CPI
  - Compilador: afeta nº de instruções, CPI
  - ISA: afeta nº de instruções, CPI, período do clock

Número Instruções como Métrica de Desempenho

- **MIPS: Millions Instructions Per Second**
- Algumas vezes usada como métrica de desempenho

  Máquinas rápidas → valor de MIPS alto
- MIPS especifica a taxa de execução das instruções

$$MIPS = \frac{\text{Nr. Instruções}}{\text{Tempo Execução} \times 10^6} = \frac{\text{Frequencia}}{\text{CPI} \times 10^6}$$

- Podemos relacionar tempo de execução a MIPS

$$\text{Tempo Execução} = \frac{\text{Nr. Inst.}}{\text{MIPS} \times 10^6} = \frac{\text{Nr. Inst.} \times \text{CPI}}{\text{Frequencia}}$$

Problemas em Usar o MIPS

- Não leva em conta a diferença de instruções

  Não é possível usar o MIPS dos computadores para comparar conjuntos de instruções diferentes, porque a contagem de instruções será diferente

- MIPS varia entre os programas no mesmo computador

  Um computador pode não ter uma classificação MIPS único para todos os programas

- MIPS pode variar inversamente com o desempenho

  Uma classificação elevada do MIPS nem sempre significa melhor desempenho

  Exemplo no próximo slide mostra esse comportamento anômalo

MIPS exemplo

- Dois compiladores diferentes estão sendo testados no mesmo programa para uma máquina de 4 GHz com três classes diferentes de instruções: Classe A, Classe B e Classe C, que exigem 1, 2 e 3 ciclos, respectivamente.

Classe	A	B	C
CPI da classe	1	2	3
Nº de Instruções compilador 1	5×10^9	1×10^9	1×10^9
Nº de Instruções compilador2	1×10^{10}	1×10^9	1×10^9

- Qual compilador produz código com o MIPS superior?
- Qual compilador produz código com o melhor tempo de execução?

Solução do MIPS exemplo

Primeiro, encontramos os ciclos de CPU para ambos os compiladores

Ciclos $_{\text{compilador1}} = (5\times1 + 1\times2 + 1\times3)\times10^9 = 10\times10^9$

Ciclos $_{\text{compilador2}} = (10\times1 + 1\times2 + 1\times3)\times10^9 = 15\times10^9$

Depois, calculamos o tempo de execução para ambos os compiladores

Tempo Execução $_{\text{compilador1}} = 10\times10^9$ cycles / 4×10^9 Hz = 2.5 sec

Tempo Execução $_{\text{compilador2}} = 15\times10^9$ cycles / 4×10^9 Hz = 3.75 sec

Compilador1 gera programa mais rápido

Agora compute a taxa MIPS para ambos compiladores

MIPS = Nr.Instruções / (Tempo Execução × 10^6)

MIPS (compilador 1) = (5+1+1) × 10^9 / (2.5 × 10^6) = 2800

MIPS (compilador 2) = (10+1+1) × 10^9 / (3.75 × 10^6) = 3200

Compilador2 gera programa com MIPS mais elevado

Otimizando Desempenho Através de Algoritmo: Arrays x Ponteiros

```
int clear1(int array[], int size)
{
  int i;
  for (i = 0; i < size; i = i + 1)
    array[i] = 0;
}
```

```
   addi $t0,$zero,0 # i = 0
L1:sll $t1,$t0,2    # $t1 = i * 4
   add $t2,$a0,$t1 #$t2= &array[i]
   sw $zero,0($t2) # array[i] = 0
   addi $t0,$t0,1   # i = i + 1
   slt $t3,$t0,$a1 # $t3=(i< size)
   bne $t3,$zero,L1
```

```
int clear2(int *array, int size) {
  int *p;
  for(p= &array[0];p< &array[size];
p = p + 1)
    *p = 0;
}
```

```
   addi $t0,$a0,0 # p = & array[0]
   sll $t1,$a1,2  # $t1 = size * 4
   add $t2,$a0,$t1 #$t2=&array[size]
L2:sw $zero,0($t0) # *p = 0
   addi $t0,$t0,4   # p = p + 1 *4
   slt $t3,$t0,$t2 #$t3=
                   #(p<&array[size])
   bne $t3,$zero,L2
```

- Versão com array requer que shift seja dentro do loop
  - Para calcular endereço do elemento do índice i

Centro de Informática

Arrays x Ponteiros

- Embora operações envolvendo arrays possam ser feitas com ponteiros, a escolha de uma forma de trabalhar ou outra pode ter impacto no desempenho

- Indexação de um array envolve:
  - Multiplicar índice pelo tamanho do tipo
  - Adicionar este valor ao endereço base do array para descobrir endereço do elemento indexado

- Ponteiros correspondem diretamente aos endereços de memória
  - Evita a complexidade extra da indexação

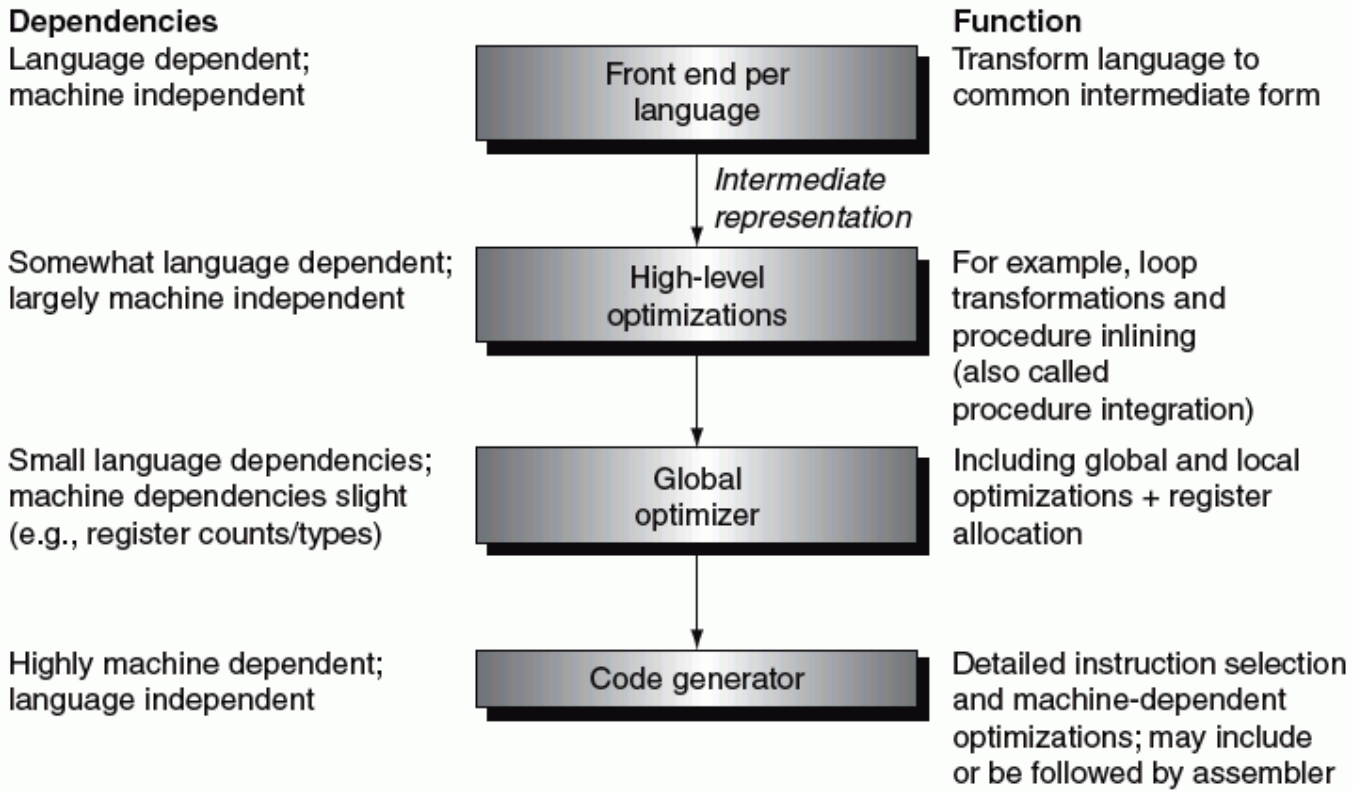
Arrays x Ponteiros

- Embora operações envolvendo arrays possam ser feitas com ponteiros, a escolha de uma forma de trabalhar ou outra pode ter impacto no desempenho

- Indexação de um array envolve:
  - Multiplicar índice pelo tamanho do tipo
  - Adicionar este valor ao endereço base do array para descobrir endereço do elemento indexado

- Ponteiros correspondem diretamente aos endereços de memória
  - Evita a complexidade extra da indexação

Otimizando Desempenho Através do Compilador



- **Diferentes tipos de otimização:**

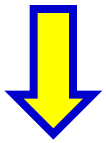
  Alto nível, local, global

Exemplo de Otimização de Alto Nível

- **Loop Unrolling**

  Consiste em replicar corpo do laço para reduzir número de iterações, evitando testes de fim de laço e jumps

```
for( i = 0; i < 10; i++){
        n = n + 5;
}
```



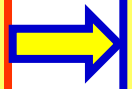
```
for( i = 0; i < 5; i++){
        n = n + 5;
        n = n + 5;
}
```

Outro Exemplo de Otimização de Alto Nível

- **Procedure Inlining**

  Consiste em substituir chamadas a procedimento pelo corpo do procedimento, para reduzir overhead da chamada

```
void imprimeSituacao(float m) {
  if (m >= 7)
    printf("\nAprovado");
  else
    printf("\nReprovado");
}
int main() {
  float media;
  printf("Digite media: ");
  scanf("%f",&media);
  imprimeSituacao(media);
}
```



```
int main() {
  float media;
  printf("Digite media: ");
  scanf("%f",&media);
  if (media >= 7)
    printf("\nAprovado");
  else
    printf("\nReprovado");

}
```

Otimização Local/Global

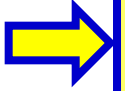
- Otimizações locais trabalham em cima de blocos simples individualmente
  - Ex: branches, expressões, loops, etc
- Otimizações globais trabalham sobre múltiplos blocos simples espalhados pelo código
- Exemplos de técnicas:
  - Strength Reduction
    - Substitui operações complexas por operações mais simples
    - Ex: Substitui instruções mult em assembly por sll quando for possível
  - Dead code elimination
    - Elimina código que não afeta resultado final do programa
    - Exs: variáveis não utilizadas, condições inalcansáveis de ifs

Exemplo de Otimização Local/Global

- **Code Motion**

  Consiste em identificar trechos de código dentro de um loop que calculam sempre o mesmo valor em todas as iterações (invariante do loop) e colocar estes trechos para fora do loop

```
int y;
int n = 0,x = 5;
for( i = 0; i < 10; i++){
      y = x + 10;
      n = n + 5;
}
```



```
int y;
int n = 0,x = 5;
y = x + 10;
for( i = 0; i < 10; i++){
      n = n + 5;
}
```

Examinando Como Diferentes Fatores Podem Influenciar Desempenho

- **Exemplo: Ordenação**

  Algoritmos : Bubble sort e Quicksort

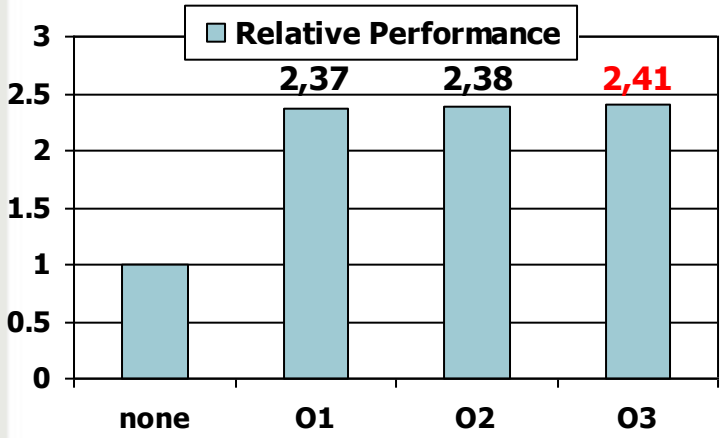
  Linguagem: C e Java

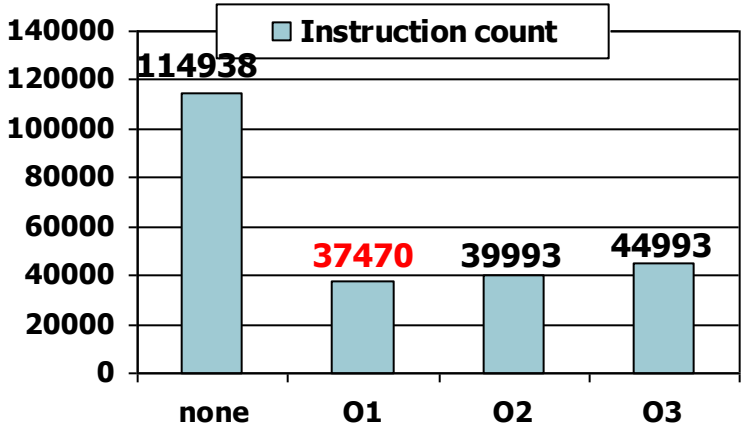
  Compilador: gcc com 3 níveis de otimização para C (O1, O2 e O3) e interpretador e JIT para Java

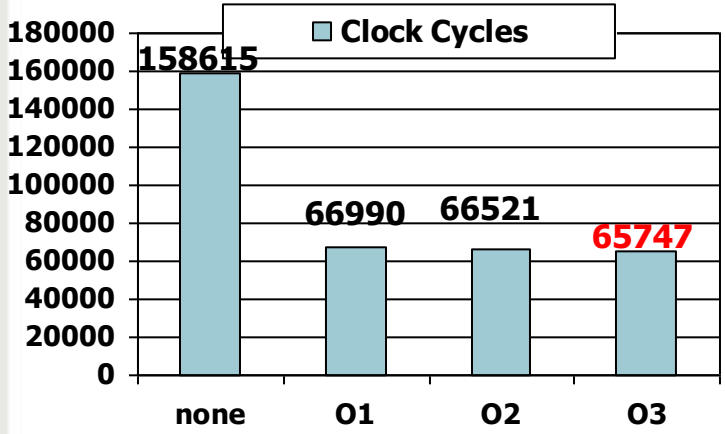
  ISA: Pentium 4 (com Linux)

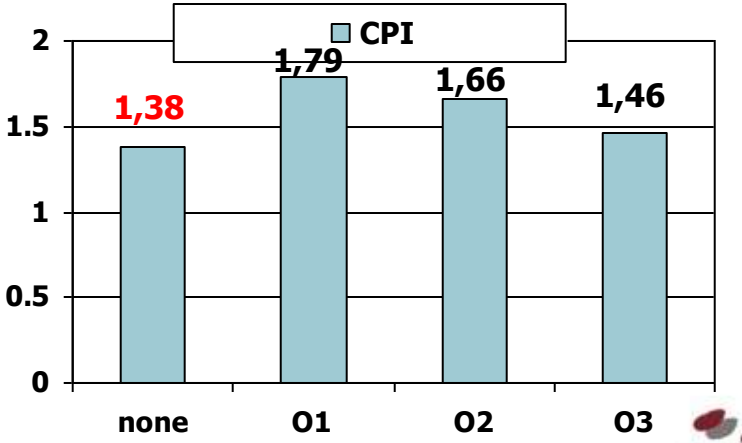
- **Comparação será feita em relação a codigos C não otimizados**

Efeito de Otimização do Compilador *gcc*



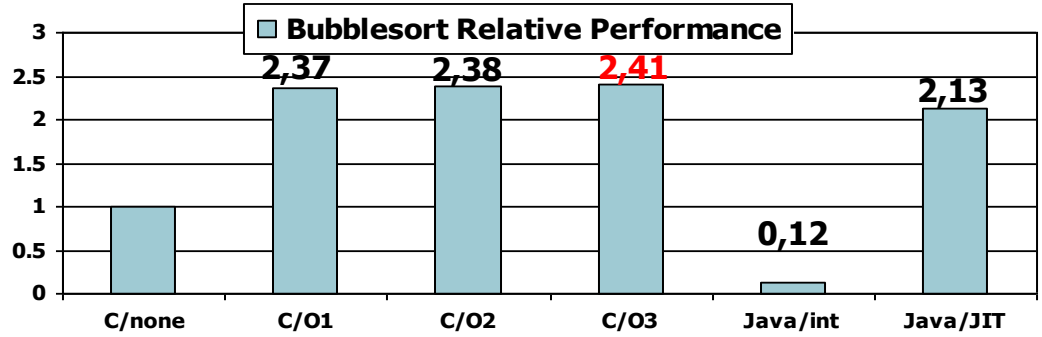


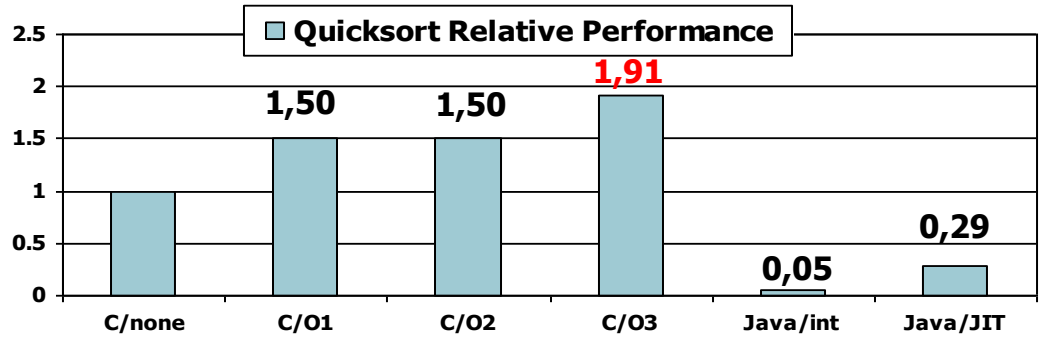


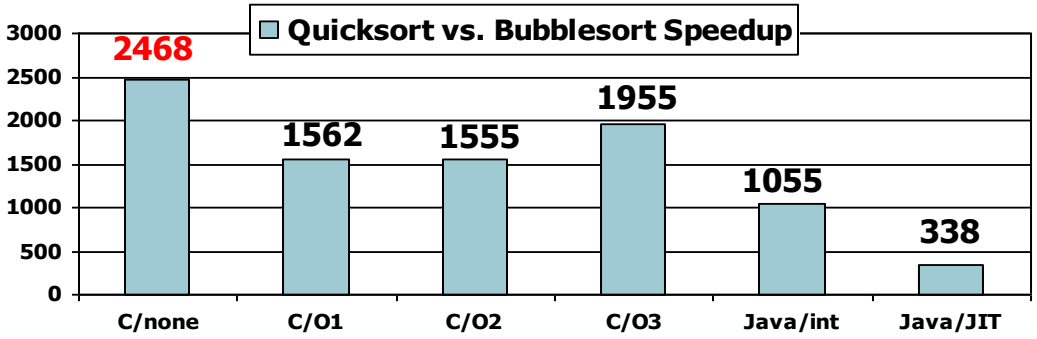


Centro de Informática

Efeito da Linguagem e Algoritmo







Analisando Resultados

- Número de instruções e CPI isoladamente não são bons indicadores de desempenho
- Otimizações de compilador são sensíveis ao algoritmo
- Otimizações nem sempre levam a um melhor desempenho
  - Pode aumentar tamanho do código, o que pode afetar desempenho, pois código pode ficar grande demais para caber em cache **(veremos depois)**
- Código compilado Java/JIT é significamente mais rápido que código interpretado pelo JVM
  - Comparável a C otimizado em alguns casos
- **Nenhuma otimização é capaz de compensar um algoritmo ruim!**

Como Podem Ser Comparados Differentes Computadores?

- Benchmarks

  Conjunto de aplicações padrões que são utilizados para avaliar o desempenho

  Vários domínios:

  - Científicos
  - Banco de dados
  - Processamento de sinais
  - Rede

- SPEC (System Performance Evaluation Cooperative)

  Iniciativa criada pela indústria para criar benchmarks para avaliar seus computadores

Benchmarks da SPEC

- Benchmarks usando inteiros

  Escritos em C e C++

- Benchmarks usando ponto flutuante

  Escritos em C e Fortran

Integer benchmarks		FP benchmarks	
Name	**Description**	**Name**	**Type**
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The Gnu C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptic partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition using neural networks
perlbmk	perl application	equake	Seismic wave propagation simulation
gap	Group theory, interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ammp	Computational chemistry
bzip2	Compression	lucas	Primality testing
twolf	Place and rote simulator	fma3d	Crash simulation using finite-element method
		sixtrack	High-energy nuclear physics accelerator design
		apsi	Meteorology: pollutant distribution