CAP-387(2016) – Tópicos Especiais em Computação Aplicada: Construção de Aplicações Massivamente Paralelas

Aula 12: Lei de Moore e Pipelining

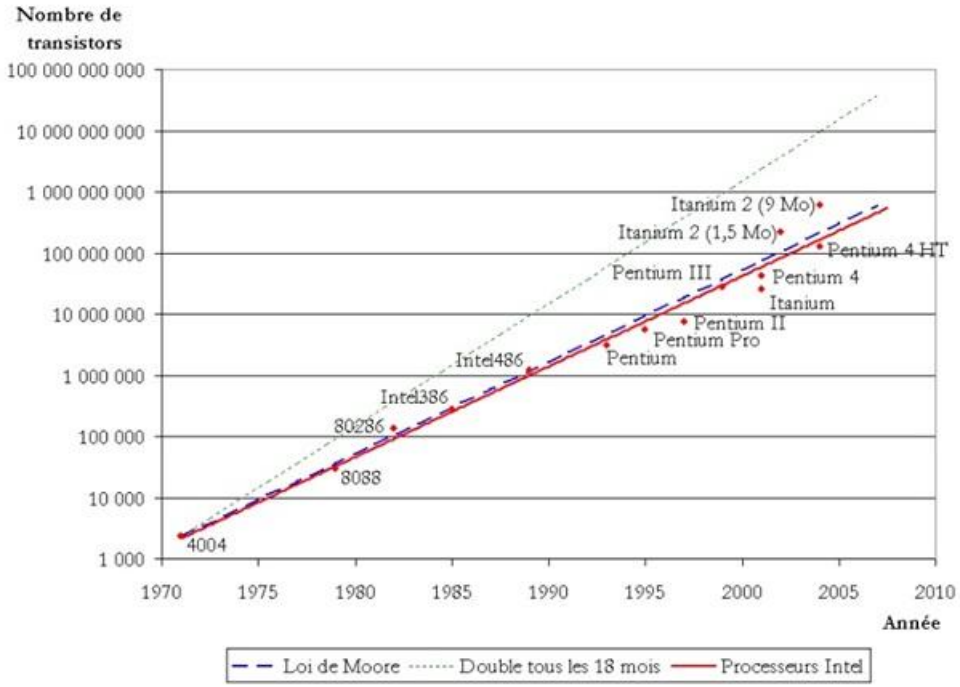
Celso L. Mendes, Stephan Stephany

LAC / INPE

Emails: celso.mendes@inpe.br, stephan.stephany@inpe.br

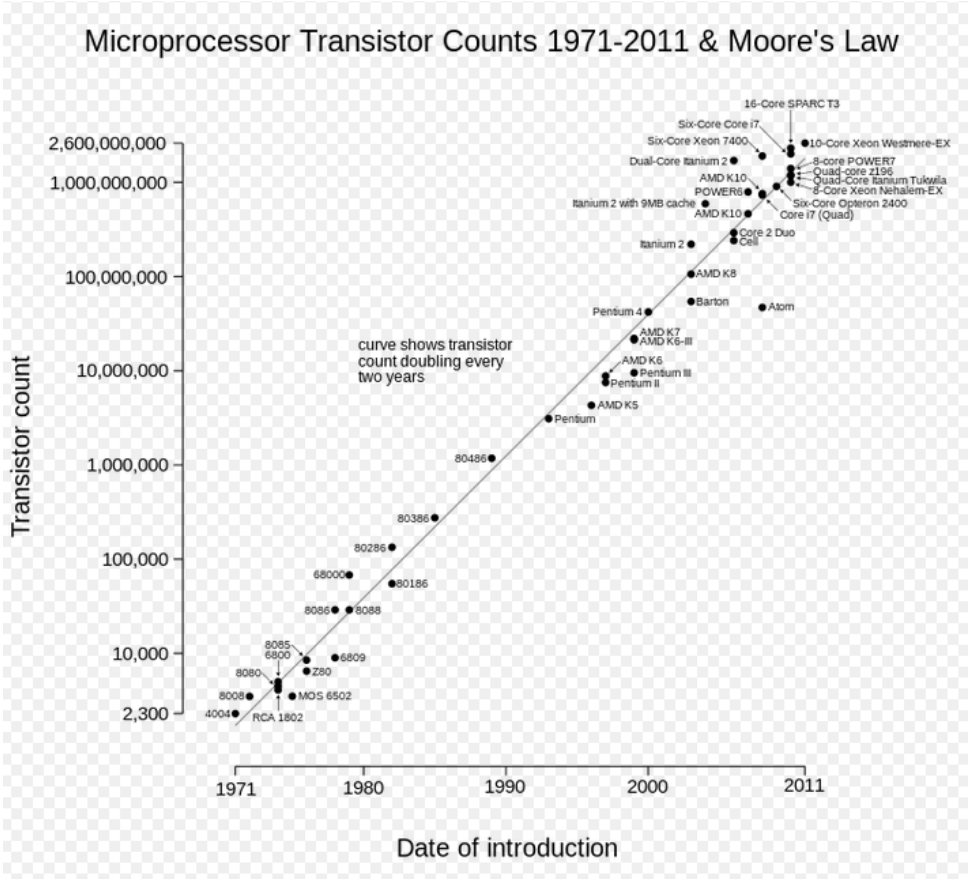
Lei de Moore

- Projeção feita na década de 60:
  - Densidade de componentes dobra a cada dois anos!
    - Percepção popular: dobro de velocidade nos processadores



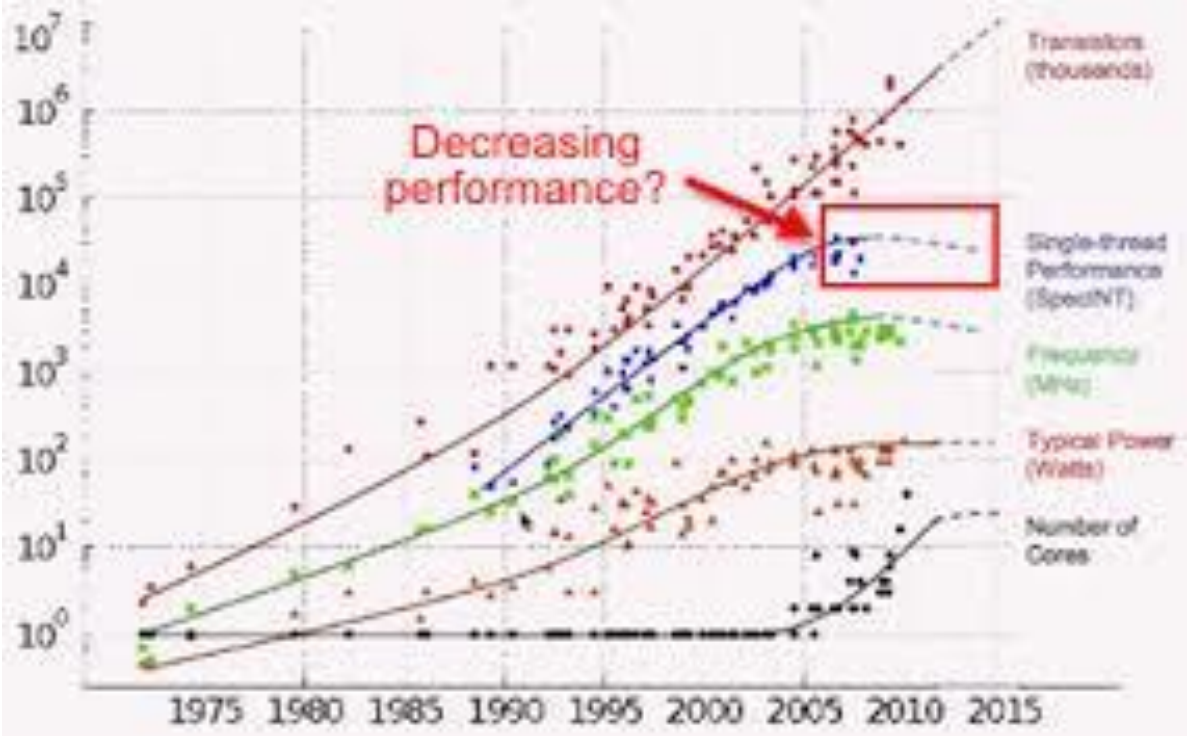
Lei de Moore

- Verificada em 4 décadas de microprocessadores!



Lei de Moore (cont.)

- Problema: energia α frequência^2
  - 2004: Manter os aumentos de frequência de antes é proibitivo!



Número de Transistores (lei de Moore)

Desempenho por Núcleo

Frequência

Consumo

Número de Núcleos

Lei de Moore (cont.)

- Conclusões:
  - Lei deMoore continua valendo até hoje!
    - Densidade de componentes (número de transistores por área de pastilha) continua dobrando a cada dois anos
    - Frequência de relógio parou de aumentar por causa da energia consumida, e do calor dissipado
  - Novos transistores são usados para replicar os núcleos dos processadores
    - Início da era *multi-core*: múltiplos núcleos por pastilha
    - Desafio: como aproveitar os vários núcleos?

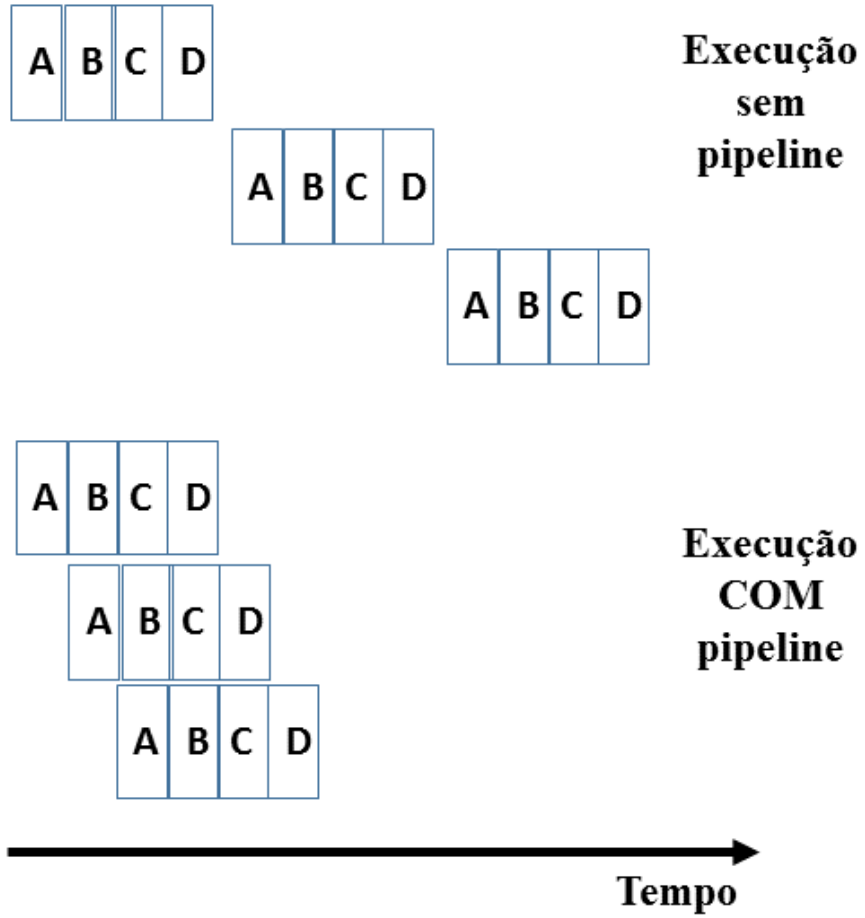
Pipeline

- Princípio Básico:
  - Aumento de eficiência via linha de montagem tradicional
    - Idéia: quebrar tarefa complexa em sub-tarefas simples
    - Executar sub-tarefas em sequência
    - Cada sub-tarefa é feita por uma unidade especializada
  - Paralelismo: várias unidades em funcionamento
  - Ganho de desempenho:
    - Não é preciso esperar uma tarefa terminar totalmente para iniciar a tarefa seguinte

Pipeline (cont.)

Exemplo: Tarefa A-B-C-D

Sub-tarefas A, B, C, D



Caso Típico: Exec. Instruções

A = Busca da Instrução

B = Decodificação

C = Acesso aos Operandos

D = Execução da Operação

Pipeline (cont.)

- Observações:
  - Tempo total de cada tarefa não muda …
  - Mas o tempo total de *N* tarefas é bem menor
  - Supondo *N* tarefas, cada uma composta de *K* sub-tarefas:
    - Tempo sem pipeline: $N \times K$ ciclos
    - Tempo com pipeline: $K + (N-1)$ ciclos
    - Ganho de desempenho com pipeline: $(N \times K)/(K + N - 1)$
      - Se $N \gg K$: $K + N - 1 \approx N$ e então Ganho $\approx K$

Ganho de desempenho com pipeline ≈ Número de estágios no pipeline

Pipeline (cont.)

- Processadores atuais:
  - Pipeline usado em todas as unidades funcionais
  - Desempenho de pico de um processador é calculado assumindo que as unidades funcionais de ponto-flutuante geram um resultado por ciclo (ou dois, se há *mult/add*)
  - Cada operação em si ainda leva vários ciclos
    - Exemplo: Intel Ivy-Bridge tem pipelines de 14~19 ciclos
  - Mas com pipelining, uma operação é completada a cada ciclo
    - Efetivamente, é como se cada operação levasse 1 ciclo!

Pipeline (cont.)

- Possíveis complicações práticas:
  1. Tempos das sub-tarefas podem não ser iguais
     - Algumas sub-tarefas podem sofrer atrasos

       Exemplo: “C” pode sofrer atraso no acesso `a memória
     - Estes atrasos são conhecidos como *bolhas no pipeline*
       - Enquanto a bolha perdurar, não há processamento útil !
  2. Uma sub-tarefa pode precisar usar como entrada algum dado ainda não disponível

     Exemplo: C_i poderia precisar acessar um dado calculado por D_{i-1}
     - Soluções de hardware: (a) travar o pipeline; (b) redirecionar dado