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

Aula 5: Desempenho e Acesso à Memória

Celso L. Mendes, Stephan Stephany

LAC /INPE

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

Desempenho em Memória - Stream

- Relembrando: 4 tipos de testes no Stream
  - COPY: x(i)=y(i)
  - SCALE: x(i)=a*y(i)
  - ADD: x(i)=y(i)+z(i)
  - TRIAD: x(i)=y(i)+a*z(i)
- Foco: vetores *grandes*
  - Forçar acessos à memória principal sempre que possível, de modo a avaliar a velocidade de acesso à memória principal

Stream - Exemplos

- Valores medidos em dois laptops reais (fonte: W.Gropp)

```
Function Rate (MB/s)   Avg time     Min time     Max time
-------------------------------------------------------------
Copy:      2900.3744   0.0115       0.0110       0.0121
Scale:     2752.9018   0.0121       0.0116       0.0137
Add:       3241.4521   0.0156       0.0148       0.0188
Triad:     3265.9560   0.0151       0.0147       0.0165
------------------------------------------------------ ------
Copy:      16970.7     0.009641     0.009428     0.010048
Scale:     13321.1     0.012168     0.012011     0.012475
Add:       13147.8     0.018488     0.018254     0.019308
Triad:     13101.7     0.019142     0.018318     0.019389
```

2008

2015

Stream – Desempenho

- Evolução do Desempenho:
  - Melhoria por um fator de ~4, em 7 anos → 22% por ano
  - Mas, pela Lei de Moore:
    - Dobro de densidade dos components a cada 2 anos
    - Consequência: dobro de velocidade (desempenho) a cada 2 anos
    - Ou seja, esperado um fator de $2^{1/2}$ por ano → 41% por ano!
  - Logo, a melhoria é aproximadamente apenas metade daquilo que seria esperado pela Lei de Moore!
    - Razões ???

Stream - Desempenho

- Análise mais Detalhada - *Triad*
  - TRIAD: x(i)=y(i)+a*z(i)

2 operações de ponto flutuante por iteração: *,+

→ Em geral, implementado com uma instrução de *multiply-add*

Supondo um processador a 2 GHz: 1 ciclo = 0.5 ηs

2 loads (2×8=16 bytes): y(i),z(i)

1 store (8 bytes): x(i)

→ Total: 24 bytes de transferência

Supondo 12 GB/s de largura de banda: T(24 bytes) = 2 ns

Logo, mesmo com acesso à memória e cálculo simultâneos, o máximo desempenho será $V_{máximo}$ = 1/4 do pico!

Stream – Desempenho (cont.)

- Tupã/CPTEC: Cray XE-6
  - CPUs: 2 AMD-Opteron@2.1GHz, 2×12=24 núcleos por nó
    - Os 24 núcleos poderiam realizer um *multiply-add* ao mesmo tempo, em 1 ciclo: $(2.1\times10^{9})^{-1}$ = 0.48 ηs
  - Memória: 32 GB por nó, com largura de banda de 102.4 GB/s
    - 24 x {2 loads + 1 store}: 72 × 8 bytes = 576 bytes → 5.625 ηs
  - Logo, para execuções simultâneas em um nó, teremos por iteração:
    - $T_{floating\text{-}point}$ = 0.48 ηs
    - $T_{memória}$ = 5.625 ηs se os acessos forem à memória principal

  → Desempenho efetivo é uma ordem de grandeza pior que o pico!

Outro Exemplo

- Operação com Matrizes Esparsas
  - Hipótese: vetores *ia(n+1),ja(nnz),a(nnz),x(n),y(n)*
    → *ja*: usado para apontar para elementos não-nulos de *x*

    Possível implemntação, em Fortran:

```
offset=0
do row=1,n
        m = ia(row+1) – ia(row)
        sum=0
        do k=1,m
                sum = sum + a(offset+k) * x(ja(offset+k))
        enddo
        y(row)=sum
        offset=offset+m
enddo
```

Outro Exemplo (cont.)

- Operação com Matrizes Esparsas – núcleo central:

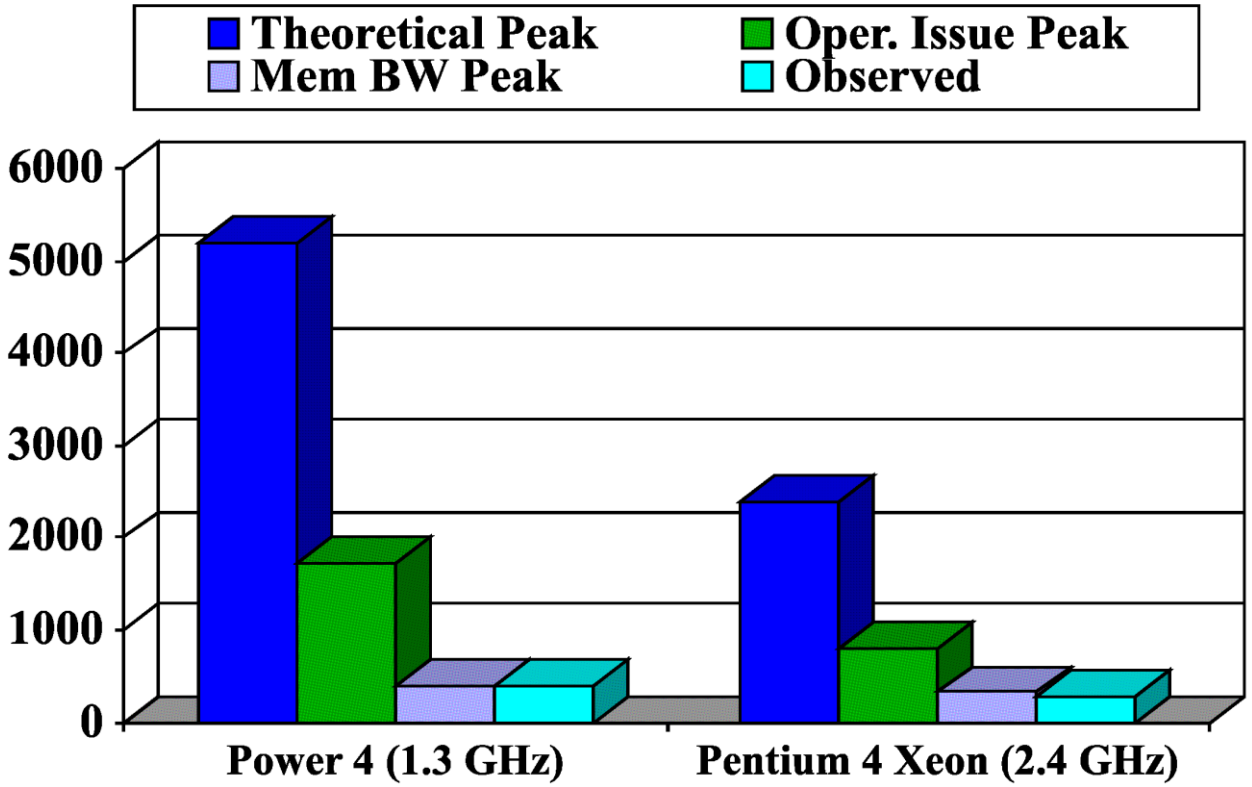
```
do k=1,m
        sum = sum + a(offset+k) * x(ja(offset+k))
enddo
```

  - Para cada iteração:
    - 2 operações de ponto-flutuante: +,*
    - Vários acessos aos arrays em memória: *a(), x(), ja()*

    (mesmo supondo que os escalares sejam mantidos em registradores)
  - Como em *Triad*, há muitas operações de memória por operação de ponto-flutuante: muitos *bytes/flop,* ou poucos *flops/byte*

Outro Exemplo (cont.)

- Operação com Matrizes Esparsas : desempenho medido



Fonte: Dinesh Kaushik, ORNL

Desempenho e Memória - Resumo

- Operações com Arrays em Memória
  - Podem limitar o desempenho final observado
  - Muitos códigos reais podem apresentar poucos *flops/byte*

  → *flop/byte: Intensidade Computacional*
    - Quantidade de cálculo (em ponto-flutuante) a ser feito por byte de dados
    - Depende do algoritmo, do compilador, etc.
    - Fator fundamental no desempenho final de um código!