SSC0951 – Desenvolvimento de Código Otimizado

1ª aula – Apresentação da disciplina e Introdução

Profa. Sarita Mazzini Bruschi

sarita@icmc.usp.br

Material baseado nos slides do curso “Análise de Aplicações e Otimização de Desempenho” – Prof. Edson Borin – Unicamp

Apresentado na ERAD-SP 2017

SSC0951 – Desenvolvimento de Código Otimizado

- Objetivos:
  - Proporcionar um aprendizado mais aprofundado sobre desenvolvido de código, focando nos requisitos de desempenho, segurança e confiabilidade

SSC0951 – Desenvolvimento de Código Otimizado

- Programa:
  - Técnicas de otimização de código: Técnicas independentes do processador, técnicas dependentes do processador, detecção de código quente, profiling: objetivos, metodologias e ferramentas.
  - Aceleração de Aplicações: Otimizações Simples; Otimizações na Compilação; Vetorização de Código; Bibliotecas Otimizadas; Otimizações de Acesso a Dados.
  - Desenvolvimento de código seguro: prevenção de buffer overflow, prevenção por formatação de strings, prevenção por overflow de inteiro.
  - Desenvolvimento de código confiável: práticas para desenvolvimento de código confiável

Bibliografia

- Hager, G. and Wellein, G. Introduction to High Performance Computing for Scientists and Engineers. CRC Press, Inc., Boca Raton, FL, USA. 2010.
- Bryant, R. and O'Hallaron, D. R. Computer Systems: A Programmer's Perspective. Prentice Hall, Third Edition, 2015.
- Viega, J; McGraw, G. Building Secure Software: How to Avoid Security Problems the Right Way. Maddison-Wesley Professional. 2001

SSC0951 – Desenvolvimento de Código Otimizado

- O curso terá aulas expositivas + exercícios
- Avaliação:
  - Trabalhos práticos que serão desenvolvidos no decorrer do curso
  - Serão de 8 a 12 trabalhos
  - Média final = média aritmética dos trabalhos práticos

Introdução

- O que é otimização?
  1. Criação de condições mais favoráveis para o desenvolvimento de algo.
  2. Estatística: processo através do qual se obtém o melhor valor de uma grandeza.
- Para nós:
  - Otimização == melhorar desempenho
  - Vamos ver também questões de segurança e confiabilidade
- Otimização de código:
  - Melhorar o desempenho do código

Desempenho

Enquanto (o desempenho não for bom suficiente)

{

Identifique os trechos de código **frequentemente executados**

Otimize o trecho de código

}

Desempenho

Enquanto (o desempenho não for bom suficiente)

{

Identifique os trechos de código **frequentemente executados**

Otimize o trecho de código

}

Dúvidas:

- Como medir o desempenho?
- Como saber se está bom o suficiente?

Desempenho

- Como medir o desempenho?
  - Utilizando uma métrica
    - Gflop/s
    - GB/s
    - MTrasactions/s
    - Frames/s
- Qual métrica usar?
  - Depende do problema!

Desempenho

- Exemplo: Multiplicação de matrizes
  - Utilizaremos Gflops/s
  - C[M][N] = A[M][K] x B[K][N]
    - # de operações de ponto flutuante:
    - M * N * K * (1 + 1)
    - flop/s = M*N*K*2 / tempo de exec.

```
for (i = 0; i < (m); i++) {
   for (j = 0; j < (n); j++) {
      t = 0;
      for (p = 0; p < (k); p++) {
         t += A[i*k + p] * B[n*p + j];
      }
      C[j+i*n] = t ;
   }
}
```

Desempenho

- Exemplo: Multiplicação de matrizes com M=1000, N=1000, e K=1000
- # de operações de ponto flutuante = M * N * K * (1 + 1) = $2x10^9$
- Supondo que o tempo de execução foi 0.5s
- Desempenho = $2x10^9$ / 0.5s = $4x10^9$ Flop/s = **4 GFlop/s**
- Como saber se já está bom o suficiente?
- Ideia: Compare com o desempenho de pico da máquina utilizada!

Desempenho

- Ideia: Compare com o desempenho de pico da máquina utilizada!
- Máquina: 2.8 GHz, 2 núcleos, instruções AVX2 (16 operações de PF)
- Total = $2.8 \times 10^9 \times 2 \times 16 = 89.6 \times 10^9$ Flop/s = 89.6 GFlops
- 89.6 GFlop/s vs 4 GFlop/s
- É muito difícil atingir o desempenho de pico!!!
- Como saber se já está bom o suficiente?

Desempenho

- Como saber se já está bom o suficiente?
- Modelo de desempenho da aplicação!
- O modelo deve levar em consideração:
  - os recursos computacionais do *hardware*: Unidades vetoriais (SSE/AVX), número de núcleos computacionais, taxa de vazão da memória, ... ; e
  - as otimizações aplicadas na aplicação: Paralelização, vetorização, ...

Desempenho

- Exemplo 2:
- Máquina: 2.8 GHz, 2 núcleos, instruções AVX2 (16 operações de PF)
- Total = 2.8 x 109 x 2 x 16 = 89.6 x 109 Flop/s = 89.6 GFlops
- 89.6 GFlop/s (Pico) vs 4 GFlop/s (Medido)
- Se a aplicação não foi vetorizada e não foi paralelizada, então o desempenho de pico deveria ser:
  - 3.3 GHz x 1 núcleo x 4 instruções de PF / ciclo = 13.2 GFlop/s
- Podemos fazer melhor?

Desempenho

- Podemos olhar para as instruções do núcleo computacional (laços da multiplicação) e computar as dependências e o paralelismo entre instruções e verificar como isso afeta o número máximo de instruções que podem ser executadas em paralelo (ILP = *Instruction Level Parallelism*).
- Supondo que o ILP seja 2:
- Desempenho: 3.3 GHz x 1 núcleo x 4 instruções de PF / ciclo = 13.2 GFlop/s

2

6.6

Desempenho

- Memória: fonte comum de gargalo de desempenho.
- Quando for este o caso, otimizações que afetam o desempenho do código no processador não causam muito efeito!
- Exemplo:
  - CPU = 2.8 x 10^9 x 2 x 16 = 89.6 x 10^9 Flop/s = 89.6 GFlops
  - Vazão da memória: 25.6 GB/s
  - Relação entre CPU e memória: 3.5 operações de PF para cada *byte* trazido da memória

Desempenho – Medindo tempo de execução

- Métricas de desempenho: X / s
  - X pode ser Flop, Transactions, Frames, ...
  - Como medir o tempo de execução da aplicação ou de trechos da aplicação (s)?
- Várias abordagens:
  - Cronômetro manual!
  - /usr/bin/time ./my_app.bin
  - gettimeofday()
  - rdtsc (retorna o TSC – *Time Stamp Counter)*
  - bibliotecas: PAPI (*Performance Application Programming Interface*), ...

Desempenho

- Outras ferramentas que podem ser utilizadas para obter informações de desempenho:
  - Gprof
  - Perf
  - Valgrind
  - Intel Vtune
  - Intel Performance Counter Monitor (PCM)

Desempenho

- Cuidado com a variabilidade e a média: diversos fatores podem afetar o tempo de execução:
  - Estruturas de *caching* do *hardware* (Processador, Hard Drive, etc...) e de *software* (sistema operacional, bibliotecas, ...)
  - Ajustes dinâmicos de frequência de operação para regular o consumo de energia (Turbo Boost, ...)
  - Outros programas sendo executados ao mesmo tempo que seu programa (*kernel* e módulos do sistema operacional, ...)
- Como garantir que os valores coletados são estatisticamente válidos?

Desempenho

Enquanto (o desempenho não for bom suficiente)

{

Identifique os trechos de código **frequentemente executados**

Otimize o trecho de código

}

Desempenho

Enquanto (o desempenho não for bom suficiente)

{

Identifique os trechos de código **frequentemente executados**

Otimize o trecho de código

}

Detecção de código quente

Desempenho

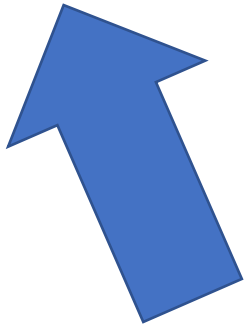
Enquanto (o desempenho não for bom suficiente)

{

Identifique os trechos de código **frequentemente executados**

Otimize o trecho de código

}



Técnicas de Otimização