Avaliação de Desempenho

Programação Concorrente e Paralela

Noemi Rodriguez

2016

O que é desempenho?

- em primeiro lugar, uma ótima tradução para *performance*... :-)
- tempo de execução (o centro das atenções!)
- outras: projeto, ciclo de vida, manutenção, ...
- mesmo outras medidas de execução podem ser importantes:
  - utilização de memória
  - throughput
  - uso da rede

Como estudar desempenho?

- extrapolação a partir de observações
  - *"implementamos o algoritmo na máquina X e obtivemos uma aceleração de 10.8 em 10 processadores"*
- análise assintótica
  - análise mostra que o tempo será O(n log n)
- mas o que está acontecendo nos casos que realmente nos interessam?
  - e que casos são esses?

Como estudar desempenho? (cont)

- modelos de desempenho
- experimentos
- simulação

Modelos de desempenho

- objetivo: explicar dados observados e prever comportamento em circunstâncias futuras
  - necessidade de abstrair detalhes menos importantes
- previsão do tempo de execução:

$$T = f(N, P, U, ...)$$

  - N: tamanho do problema
  - P: número de processadores
  - U: número de tarefas
  - ... outras características

Definições

tempo de execução

- tempo decorrido do momento em que o primeiro processador começa a executar uma tarefa da aplicação até o momento em que o último processador para de executar.
  - tb chamado de *wall clock time*

paralelismo: tentativa de modelo

- podemos estimar o tempo em paralelo por:

$$T_{paralelo} = T_{serial}/p + T_{sobrecarga}$$

- ... supondo que o trabalho serial foi dividido exatamente pelos processadores

outro modelo

- podemos olhar o tempo em cada processador:

$$T = T^j_{comp} + T^j_{comm} + T^j_{idle}$$

- ou o tempo total:

$$T = (T_{comp} + T_{comm} + T_{idle})/P$$

- que tempo (ou gasto de recursos) realmente nos importa?

Reduzindo complexidade

- desenvolver uma expressão matemática para descrever T é uma tarefa complexa...
- *máquina ideal*
  - sem preocupação com topologia da rede, hierarquia de memória, etc
- análise em escala
  - tentativa de identificar fatores insignificantes
- análise empírica
  - calibragem de modelo com experimentos

Tempo de computação

o que medir?

- flops (número de operações de ponto flutuante por segundo)?
  - críticas...
- tempo de CPU
- ???

- possibilidade de medir partes em programa sequencial
- implementação de *kernels* para medidas
- cuidados com alterações devidas a memória, etc

Tempo de comunicação

- diferença intra e inter-máquinas
- tempo idealizado para mensagens:

$$T_{msg} = t_{startup} + t_{tword} L$$

- experimentos específicos podem determinar esses tempos
- tempo de sincronização – como medir?

aceleração

- aceleração (*speedup*) – ganho com P processadores

$$A_{relativa} = \frac{T_1}{T_P}$$

eficiência

- eficiência (efficiency) – utilização de cada processador

$$E_{relativa} = \frac{T_1}{PT_P}$$

como medir o tempo sequencial?

aceleração e eficiência absolutas

- tempo do melhor algoritmo sequencial

Aceleração

lei de Amdahl [a]

[a] G. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. AFIPS '67. ACM, 483-485.

- se programa tem fração $1/s$ inerentemente sequencial, a maior aceleração que conseguiremos é de s

Modelo com lei de Amdahl

$$T_{paralelo} = T_{serialparalelizável}/p + T_{serialnãoparalelizável} + T_{sobrecarga}$$

- se $s = 0,1$ e $T_{serial} = 20$ e $T_{sobrecarga} = 0$

Modelo com lei de Amdahl

$$T_{paralelo} = T_{serialparalelizável}/p + T_{serialnãoparalelizável} + T_{sobrecarga}$$

- supondo $T_{serialnãoparalelizável} = 0$ e $T_{sobrecarga} = 0$, conseguimos $T_{paralelo} = T_{serial}/p$:
  - $A = T_{serial}/T_{paralelo} = p$
  - $E = A/p = 1$ aceleração ideal!

Anomalias em aceleração

- aceleração anômala ou super linear
- motivos:
  - cache, memória virtual
  - irregularidade das estruturas do problema

Escalabilidade

- algoritmo pode ser adaptado?
- como reage a crescimento de N?
- como reage a crescimento de P?

Escalabilidade com N fixo

- qual o maior número de processadores que podem ser usados produtivamente?

Escalabilidade com N crescendo

- qual o maior número de processadores que podem ser usados produtivamente?
- conceito de *isoeficiência*:
  - como a quantidade de computação tem que crescer, quando P cresce, para manter a eficiência constante?
  - $E = f(n, p)$ – se multiplicarmos p por k, que fator temos que aplicar a n?
    - exemplo: $E = n/(n + p)$

Estudo experimental

- abordagem iterativa
  1. experimentos para encontrar parâmetros ($t_{startup}$, etc)
  2. análise teórica
  3. implementação
  4. experimentos para confimar previsões de análise

medições

caráter de experimentos pode ser bem distinto nas diferentes fases

Projeto de experimentos

- área com suas próprias questões e literatura
- objetivos dos experimentos
  - twelve ways to fool the masses when giving performance results on parallel computers (David Bailey)
- reprodução de experimentos já realizados por outros grupos
  - quase sempre uma dificuldade!

Projeto sistemático de experimentos

- definição precisa de objetivos (fronteiras)
- seleção de métricas
- enumeração de parâmetros que afetam o desempenho
  - podem alterar o resultado mas não são necessariamente de interesse de estudo
- seleção de fatores para estudo
  - podem alterar o resultado e queremos saber como
- seleção de carga de trabalho
- projetos dos experimentos
- análise e interpretação de resultados

Métricas

- velocidade
  - tempo de resposta
  - *throughput*
  - recursos consumidos
- confiabilidade
  - tempo entre falhas
- disponibilidade
  - fração do tempo em que sistema está disponível
- outros que não sabemos como medir...
  - usabilidade
  - flexibilidade
  - ...

Cargas de Trabalho

- históricas X sintéticas
- nível (aplicação, sistema operacional, CPU)
- representatividade (relação da carga sintética com a carga real...)
  - taxa de chegada de pedidos
  - demanda de recursos
  - perfil de utilização

Avaliação – motivos para surpresas

- desbalanceamentos de carga
- computação replicada
- algoritmo e ferramenta que não combinam
- competição por banda passante

Motivos para surpresas – programação sequencial

- compiladores são cada vez mais espertos mas...
- otimizações podem fazer muita diferença na execução de um programa (compilador não tem como fazer algumas substituições!)
  - testes dentro ou fora de loops

```
...
for (i=0; i<vec_length(v); i++) {
  data_t val;
  get_vec_element (v, i, &val);
  ...
 }
```

    - e, claro, testes desnecessários!
  - acessos desnecessários a memória

```
void twiddle1 (int *xp, int *yp) {
  *xp += *yp; *xp += *yp;
}
void twiddle2 (int *xp, int *yp) {
  *xp += 2* *yp;
}
```

Motivos para surpresas – cache e programação paralela

- importância do cache e problema de coerência
  - acessos a variáveis compartilhadas, mesmo de leitura, podem ficar caras por causarem acessos à memória principal
- falso compartilhamento
  - verificações são feitas por *linha* de cache
  - exemplo multiplicação matriz por vetor: 8.000X8.000, 8X8.000.000

Medidas de Tempos

- que partes do programa queremos contabilizar?
  - normalmente concentração em trechos de processamento mais pesado
- que tipo de medida de tempo utilizar?
  - wall-clock time X tempo de CPU
  - wall-clock time:

```
start = getcurrenttime();
... trabalha
finish = getcurrenttime();
```

- precisão do timer e uso de repetições
- em que linha de execução medir?

Experimentos e Medidas – Variação de resultados

- algoritmos não determinísticos
- condições de execução (sistema operacional, memória, etc)
- custos de inicialização e terminação
- interferência de outros programas
- interferência entre experimentos
  - executar em diferentes ordens pode explicitar interferência
- alocação de recursos aleatória

- o que relatar?
  - citar variância
  - média, melhor caso, pior caso?

Como medir tempo

diferentes usos

- `clock`
- `time`
- `getcurrenttime`

cuidados

- resolução
- foco na atividade a ser medida

Referências

- David H. Bailey. Misleading performance claims in parallel computations. In *Proc. of the 46th Annual Design Automation Conference (DAC '09)*, 528–533. New York, USA. 2009. ler!!
- Ian Foster. Designing and Building Parallel Programs. Addison-Wesley, 1995. Disponível em `mcs.anl.gov/~itf/dbpp/`. ler cap. 3!!
- R. Jain. The Art of Computer Systems Performance Analysis. Wiley, 1991.
- Peter Pacheco. An Introduction to Parallel Programming. Elsevier, 2011.