

# Programação Concorrente e Paralela

Noemi Rodriguez

August 19, 2009

- princípios e técnicas de programação paralela
  - multiprocessadores
    - memória compartilhada
    - troca de mensagens
    - arquiteturas alternativas
  - multicomputadores
    - troca de mensagens

*obs:* Essa troca de mensagens pode ser vista com diferentes níveis de abstração!

# que princípios e técnicas são esses?

- notações para controle de fluxo
  - criação de processos e threads, ou outras formas de expressão de paralelismo
- abstrações de comunicação
- sincronização em sistemas de memória compartilhada
- propriedades: *safety* e *liveness*
- avaliação de desempenho
- balanceamento de carga

- G. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 2000.
- M. Herlihy, N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2008.
- M. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2003.
- artigos técnicos.

- aplicações que em princípio poderiam ser executadas de forma sequencial
- paralelismo para melhoria de desempenho ou viabilidade de execução
  - lei de Amdahl

- anos 60 e 70: máquinas construídas especificamente para paralelismo
  - alto custo X baixa procura
  - uso principalmente para computação científica
- anos 80: redes de estações de trabalho
  - aproveitamento de recursos ociosos
  - heterogeneidade
- anos 90: clusters
  - baixo custo (inicialmente)
  - homogeneidade
- anos 90 e 00: grades
  - distribuição geográfica
  - ambientes altamente heterogêneos
- anos 00: multicores e multiprocessadores “off the shelf”

- SISD
  - single instruction single data
- SIMD
  - single instruction multiple data
  - extensão para SPMD
- MISD
  - multiple instruction single data
- MIMD
  - multiple instruction multiple data

- processos e threads
  - pilha de execução
  - dados globais
  - espaços de endereçamento protegidos

# Exemplo: Busca de primos com memória compartilhada

- imprimir primos entre 1 e  $10^{10}$
- máquina de 10 processadores
- um thread por processador

# Busca de Primos com Memória Compartilhada

- idéia 1: cada thread testa um intervalo pré-definido
- cada thread executa:

```
void primePrint {  
    int i = ThreadID.get(); // IDs in {0..9}  
    for (j = i*109+1, j<(i+1)*109; j++) {  
        if (isPrime(j))  
            print(j);  
    }  
}
```

- nem todos os intervalos têm números iguais de primos
- números grandes: maior dificuldade computacional
- desbalanceamento

# Busca de Primos com Memória Compartilhada

- idéia 2: contador compartilhado
- cada thread executa:

```
int counter = new Counter(1);
```

```
void primePrint {  
    long j = 0;  
    while (j < 1010) {  
        j = counter.getAndIncrement();  
        if (isPrime(j))  
            print(j);  
    }  
}
```

# Contador Compartilhado

- o objeto `counter` é único para todos os threads

```
public class Counter {  
    private long value;  
  
    public long getAndIncrement() {  
        return value++;  
    }  
}
```

# Contador Compartilhado

- o objeto `counter` é único para todos os threads
- e o problema é que:

```
public class Counter {  
    private long value;  
  
    public long getAndIncrement() {  
        return value++;  
    }  
}
```

The diagram illustrates a race condition in the `getAndIncrement()` method. It shows the state of the counter before the increment operation. The code snippet shows the method returning `value++`. An arrow points from this line to a box containing the following code:

```
temp = value;  
value = value + 1;  
return temp
```

- necessidade de execução atômica e *exclusão mútua*

- slides livro Herlihy
- propriedades de *safety* e *liveness*

# Multiplicação de matrizes com ou sem memória compartilhada

- *embarrassingly* parallel application
  - com memória compartilhada: um thread por *chunk* de resultados
  - sem memória compartilhada: replicação de matriz B
  - problema da granularidade
- e se quisermos achar potências de uma matriz?