..... TASLAK .....

# Koşut İşlemler ve Eş Zamanlı Programlama

## (Parallelism and Concurrency)

## *D'nin ve C++11'in sundukları olanaklar*

**Ali Çehreli**

30 Haziran 2012; Tütev, Ankara

..... TASLAK .....

# Donanımdaki Değişiklikler

Mikroişlemci tasarımları fiziğin sınırındalar.

- Fazla büyük olduğunda bir uçtan diğer uca iletişim zaman alıyor
- 4 GHz hızdan daha fazlası pratik değil; daha yüksek hızlar fazla ısı üretiyor

Çözüm olarak hız değil, işlem birimi sayısı artıyor:

- Çok çekirdekli işlemciler
- Çok işlemcili bilgisayarlar
- GPU'lu bilgisayarlar
- Aynı binada birden fazla bilgisayar
- Bulut üzerinde binlerce işlemci

..... TASLAK .....

# Yazılımda Gereken Değişiklikler

Programcılar farklı düşünmeye başlamak zorundalar.

- Moore'un yasası ancak 5-10 sene daha devam edecek.
- Programlarımızın hangi bölümlerinin yerel veya büyük ölçeklerde koşut olarak işletilebileceklerini belirlememiz gerekiyor.
- Bugünden koşut olarak tasarlanmış olan programlar donanım geliştikçe bedavaya konmaya devam edecekler.

Daha fazla bilgi için Herb Sutter'ın iki makalesi:

- The Free Lunch Is Over
- Welcome to the Jungle

..... TASLAK .....

# Koşutluk ve Tanımlar

Alt düzeyden üst düzeye doğru:

- Mikroişlemcinin koşutluğu (pipelining, instruction reordering)
- İş parçacığı (thread)

```
a = 1;  |    b = 2;
c = 3;  |    d = a;
```

Karışık sırada işletilebilir:

```
a = 1; b = 2; d = a; c = 3;
```

- Eş zamanlı programlama (concurrency)
- Koşut işlemler (parallelism)

..... TASLAK .....

# C++03 Bellek Modelinin Yetersizlikleri

- İşlem sıralarının belirsizliği. Dekker'in örneği:

```
// Başlangıçta bütün değişkenler 0 olsun
 x  = 1;  |    y = 1;
 r1 = y;  |    r2 = x;
```

Soru: r1==0 ve r2==0 olabilir mi?

..... TASLAK .....

# Atomik İşlemlerin Gerekliliği

Hans Boehm'ün "Threads and Shared Variables in C++11" sunumundan örnek:

```
    x = 300;  |    x = 100;
```

Baytlara farklı anlarda yazılıyorsa:

```
x_high = 0;
x_high = 1;  // x = 256
x_low =  44; // x = 300;
x_low = 100; // x = 356;
```

# İşlem Sıralarının Önemi

```
x = 42;          |    while (!done) {}
done = true;     |        assert(x == 42);
```

..... TASLAK .....

# *Data Race* Tanımı

- İki bellek erişiminin çakışma halinde olması: Aynı *skaler nesneye* (örneğin değişkene) erişiyorlarsa ve en az birisi yazma erişimiyse. (Örnek: x = 1; and r2 = x; çakışır)
- *Data race* halinde olmaları:
  - Eğer çakışma halindelerse
  - ve aynı ana rastlama olasılıkları varsa

C++11 yalnızca *data race* halinde olmadıkları durum için garanti getirir.

# C++11'in Getirdiği Garanti

- atomic<T>
- atomic_...

```
atomic<int> x,y;

x = 1;          |   y = 1;
r1 = y;      |   r2 = x;
```

```
atomic<bool> done;// initially false

x = 42;       |    while (!done) {}
done = true; |        assert(x == 42);
```

Artık sıralama belirsizliği yok.