Slides to Accompany *Programming Languages and Methodologies*

R. J. Schalkoff

Chapter 15, Part 1: Parallel Programming

Parallel Programming: Algorithms and Implementation

- Key issues of parallel programming:
  - parallel algorithm decomposition
  - parallel hardware development (or availability)
- A very real and significant problem is *how to (perhaps automatically) decompose a given processing task into one which may be executed in parallel segments.*

Questions

1. Where is the parallelism in the desired computation?
2. How do I articulate/exploit it in my programming language?

The answers usually require some consideration of the underlying computational architecture.

A Simple `minic` Example

Consider the simple `minic` code fragment:

```
a=10;
b=sqrt(a);
c=a+b;
d=6;
e=sqrt(d);
f=d+e;
g=f+c;
```

Figure 1: Simple `minic` Code Fragment for Potential Parallelism Analysis

In this example, it is reasonably easy to (informally) identify parts of this code which are suitable for parallel execution. For example, two blocks:

```
a=10;
b=sqrt(a);
c=a+b;
```

```
d=6;
e=sqrt(d);
f=d+e;
```

could be computed concurrently. However, the computation required by

```
g=f+c;
```

must necessarily follow these computations.

Example: Control Constructs and Parallelism

Consider the code fragment:

```
b=sqrt(a);
if (b<4) then
    if (a==1) b=4;
c=a+b;
d=6;
if (c>a) then d=1;
e=sqrt(d);
if (e>a) then e=1;
f=d+e;
g=f+c;
```

Figure 2: `minic` Code Fragment With Conditional Constructs

Clearly identifying parallelism in this case is more challenging.

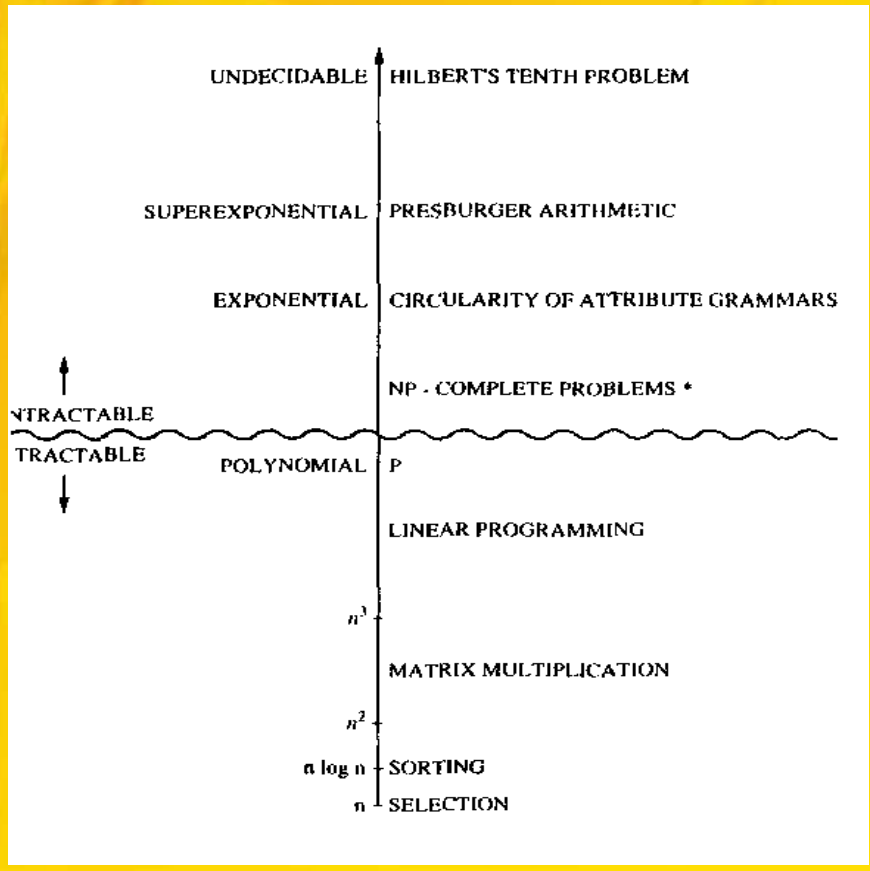

Figure 3: The Spectrum of Computational Complexity

Parallel Performance Limitations

- Intuitively, one might hope that an n processor implementation of an algorithm or parallelized program will achieve the result in $1/n$ (or less) of the time required by a single processor working on the same problem.
- Unfortunately, this is at best an upper bound on the achievable performance.

Metrics for Speedup

For a parallel implementation, *speedup* is the ratio:

$$speedup = \frac{\text{processing time with single processor}}{\text{processing time with } n \text{ processors}} = \frac{t_s}{t_p} \qquad (1)$$

On this basis, the theoretically achievable maximum speedup for an n-processor implementation of an algorithm is n.

Amdahl's Law

Amdahl's law explicitly predicts speedup as a function of the fraction of required serial computations in a given algorithm.

Define the following quantities:

s**:** the fraction of computations which are necessarily serial;

p : the fraction of computations which may be done in parallel (note $s + p = 1$); and

n : number of processors over which p is distributed.

Amdahl's law predicts speedup as:

$$speedup = \frac{(s+p)}{(s+p/n)} = \frac{1}{(s+p/n)} \tag{2}$$

In general, Amdahl's law is a pessimistic bound on achievable speedup, even for small values of s ($< 5\%$).