Special Issue
-- Parallel Computing
Sequential Programming

Traditionally, computer software has been written for *serial* computation.

- An algorithm is constructed and implemented as a serial stream of instructions.
- These instructions are executed on a central processing unit on one computer.
- Only one instruction may execute at a time.
Type of Parallelism

Several different forms of parallel computing are developed:

- Bit level: e.g. 4-bit\(^1\), 8-bit\(^2\), 16-bit\(^3\), 32-bit\(^4\), and 64-bit\(^5\) microprocessors
- Instruction level: e.g. pipeline
- Data level: e.g. SIMD
- Task level: e.g. MIMD

---

\(^1\)1970s
\(^2\)Intel 8080 in the late 1970s
\(^3\)1980s
\(^4\)Motorola MC68000, 1979
\(^5\)AMD64, aka x86-64, 2003
Example: Five-Stage Pipeline

<table>
<thead>
<tr>
<th></th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

$i \quad t$
Amdahl’s Law (1967) states that the speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.

- Let $T_1 = 1$ and $T_2 > 0$ be the running times before and after parallelization.
- Let $\alpha$ be the fraction of running time a program spends on non-parallelizable parts and $P$ be the number of processors used in parallel.

---

6Gene Myron Amdahl (1922-)
Then the speedup is given by

\[
\frac{T_1}{T_2} = \frac{1}{1 - \frac{\alpha}{P}} + \alpha \rightarrow \frac{1}{\alpha},
\]

after taking \( P \rightarrow \infty \).

Understanding data dependencies is fundamental in implementing parallel algorithms.

Besides, overhead caused by parallellism degrades the speedup.
Example

clear; clc;

a=[0.05 0.1 0.2 0.4 0.8];
p=2.^[1 2 3 4 5 6 7 8];
colorSet='rgbyk';
figure; hold on; grid on;
for i=1:length(a)
    sp=1./((1-a(i))./p+a(i));
    plot(p,sp,[colorSet(i) ':.']);
end
xlabel('Number of Processors'); ylabel('Speedup');
legend({'$\alpha=0.05$','$\alpha=0.1$','$\alpha=0.2$','$\alpha=0.4$','$\alpha=0.8$'});
print(gcf,'-depsc','Amdahl_run.eps');
Example: Speedup

```
Number of Processors

α = 0.05
α = 0.1
α = 0.2
α = 0.4
α = 0.8
```

![Graph showing Speedup vs Number of Processors for different values of α.]

- Red dashed line: α = 0.05
- Green dotted line: α = 0.1
- Light blue solid line: α = 0.2
- Yellow dotted line: α = 0.4
- Black dashed-dot line: α = 0.8

The graph illustrates how speedup changes with the number of processors for various values of α.
Example: Efficiency

![Graph showing efficiency vs. number of processors]

- $\alpha = 0.05$
- $\alpha = 0.1$
- $\alpha = 0.2$
- $\alpha = 0.4$
- $\alpha = 0.8$
MATLAB supports three kinds of parallelism: multithreaded parallelism, distributed computing, and explicit parallelism.

- Multithreaded parallelism
- Distributed computing
- Explicit parallelism

---

7 Parallel Computing Toolbox is available since 2010 version.
Interactively Run a Loop in Parallel

Using `parfor` allows you to write a loop for a statement or block of code that executes in parallel.

```
1   parfor (loopvar = initval:endval, M)
2       statements
3       end
```

- Use $M \in \mathbb{N}$ to specify the maximum number of MATLAB workers that will evaluate statements in the body of the `parfor`-loop.
- The maximum amount of data that can be transferred in a single chunk between client and workers in the execution of a `parfor`-loop is determined by the JVM\(^8\) memory allocation limit.

\(^8\)aka Java Virtual Machine
Example

- The parfor-loop works because each element depends only upon its iteration of the loop, and upon no other iterations.
- If the iterations are not independent, then they cannot be evaluated in parallel.

```matlab
clear; clc
parfor i = 1:8
    A(i)=i;
    disp(i)
end
A
```

- Yet, a simple parfor replacement is not enough.
Example

```matlab
clc; clear;
n_ = logspace(4, 8, 5);
t = zeros(length(n_), 2);
color = 'rgbk';
m = [2 4 6 8];
matlabpool 4;
figure; grid on; hold on;
for j = 1:length(m)
    for i = 1:length(n_)
        n = n_(i);
        tic;
        forTest(n);
        t(i, 1) = toc;
        tic
        parforTest(n, m(j));
        t(i, 2) = toc;
    end
```
```matlab
plot(log10(n_), t(:,1)./t(:,2)/m(j)*100, [color(j) ... 'o:']);
end
matlabpool close;
set(gca,'fontsize',16);
xlabel('Number of Data (Log Scale)'); ...
ylabel('Efficiency (%)');
legend({'m=2','m=4','m=6','m=8'},'location','NorthWest');
print(gcf,'-depsc','./pic/parfor_example.eps');
```
The safest approach when creating a parfor-loop is to assume that iterations are performed on different MATLAB workers in the parallel pool, so there is no sharing of information between iterations.
MATLAB Distributed Computing Server runs on a distributed computing resource, such as computers in a **cluster** or virtual machines in a cloud computing service.

- Access to a specific number\(^9\) of MATLAB workers that receive and execute MATLAB code and Simulink models
- Multiple users can run their applications on the server simultaneously

---

\(^9\) MDCS licences

\(^{10}\) See [MATLAB Distributed Computing Server](#).
Cluster Computers Running MATLAB Workers

The features are summarized as follows:

- Maximum of 1 MATLAB worker per CPU core is recommended.
- Homogeneous cluster configurations are recommended.
- Clusters using third-party schedulers is supported.