Performance Analysis

- Evaluation of the cost vs benefit relative to performance requires a metric or metrics by which performance is measured.

- For the first part of the course all measures of interest are based on execution time. (throughput will be discussed later)

- We want to know how much the execution time will be reduced given an architectural change.
For existing machines, benchmark suites can be used to varying degrees of reliability

- SPEC
- Perfect
- NAS computational fluid dynamics benchmarks
- Whetstones
- Dhrystones
- customized application-specific benchmarks
• Issues:

  – Workload characterization

  – Choice of metric and averaging (see the text for a discussion)

  – many problem sizes (relative to architectural parameters)

  – multiple versions (generic, SW-tuned, Manually tuned)
• For codes or machines that do not exist prediction is needed.

• Code may not exist on a given machine due to:
  – in development
  – performance tuning (via SW or Manually)

• Simulation can be used to varying degrees of accuracy (more accurate costs more)

• performance models and basic performance data from synthetic benchmarks can be used carefully

• characterization of algorithm in terms of parameters that can be related to architectural parameters is crucial
• Wall-clock or elapsed time is the ultimate measure

• \( \text{elapsed} = \text{CPUtime} + I/O\text{time} + \text{Otherstime} \)

• \( \text{CPUtime} = \text{user} + \text{system} \)

• We are interested for now in the user component of \( \text{CPUtime} \)
\[
\frac{\text{seconds}}{\text{program}} = \frac{\text{Instructions}}{\text{program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

- \textit{seconds/cycle} is the cycle time of the machine.
  
  - currently typically in nanoseconds \((10^{-9})\)
  
  - also expressed in terms of clock rate
    1\textit{/cycle time} or cycles per second \((\text{Hz})\)
    
    - \(1/100\text{ns} = 1/100 \times 10^{-9} = 10^7\text{Hz} = 10\text{MHz}\)
  
  - reduced by improving the hardware technology
• *Instructions/program* is the instruction count.
  
  – simplistic measure of complexity
  
  – better if count per instruction included
  
  – smallest count not necessarily faster code
  
  – given a system, time may be reduced by changing instruction mix or reducing instruction count, i.e., designing a better algorithm
• *cycles/instruction* is given by the number of cycles over the number of instructions issued.

• Improving this parameter is the goal of most of our discussions.

• Done through architectural improvements for a fixed code.

• Reduced by effectively performing more operations per cycle.
  
  – better functional units
  
  – instruction parallelism
  
  – improved memory access
• Better performance is measured by smaller execution time.

• a performance metric that is larger when performance is better must therefore be

\[
\text{performance} = \frac{C}{\text{execution time}}
\]

where \(C\) is a constant with respect to the architecture.

• a faster machine \(X\) should yield a \textbf{speedup} over a slower one \(Y\)

\[
\text{Speedup} = \frac{\text{time}_Y}{\text{time}_X} = \frac{\text{performance}_X}{\text{performance}_Y}
\]
Amdahl’s Law

Suppose we have an architectural improvement that speeds up part of a computation by a factor of $S$. Let $t_s$ be the amount of time it takes to perform a computation using only the slow mode of processing. Now consider the amount of time it takes to perform the computations if the computations responsible for some fraction $\nu$ of $t_s$ can be performed in fast mode. The new execution time can be written

$$t = \nu \frac{t_s}{S} + (1 - \nu) t_s$$

The speedup over slow processing is given by

$$\frac{t_s}{t} = \frac{1}{\nu + 1 - \nu} = \frac{S}{(1 - \nu)S + \nu} = gS$$

It follows that $g$ can be viewed as the fraction of the best speedup possible using the fast mode given $\nu$. 
Unfortunately, the function \( g \) for various \( S \) shows a clear diminishing returns effect. The more benefit there is to be gained via the fast mode the larger \( \nu \) must be to achieve a given speedup.
To see how quickly this degradation occurs consider writing $\nu$ as a function of $g$ and $S$. We have $0 \leq \nu \leq 1$, $1/S \leq g \leq 1$, $S \geq 1$, and

$$g^{-1} = \nu + (1 - \nu)S = S + (1 - S)\nu.$$  

Let $g = \alpha^{-1}$, i.e., the mixed-mode computations run a factor of $\alpha$ slower than the best rate. It follows that

$$\nu(S, \alpha) = \frac{S - \alpha}{S - 1}.$$  

So consider the fraction, $\nu_{1/2}$ needed so that $g = 1/2$. It becomes close to 1 very quickly.

<table>
<thead>
<tr>
<th>S</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\nu_{1/2}$</td>
<td>0</td>
<td>.50</td>
<td>.75</td>
<td>.89</td>
<td>.95</td>
</tr>
</tbody>
</table>
Other Metrics

- Performance can be measured as a rate of computation.

- MIPS – Millions of instructions per second

\[
MIPS = \frac{\text{instruction count}}{\text{time} \times 10^6}
\]

- MFLOPS – Millions of floating point operations per second

\[
MFLOPS = \frac{\text{floating point count}}{\text{time} \times 10^6}
\]
• Instructions differ in complexity from one machine to another so MIPS is not normalized

• MFLOPS does not have this problem

• Higher MFLOPS or MIPS does not necessarily translate into higher performance if the algorithm and code generation system are not fixed.
  
  – different compilers may do different levels of optimization
  
  – problem structure may be exploitable to reduce instruction count