Computing Performance

- Program execution time
  - Elapsed time from program initiation and input to program output
  - Depends on hardware, machine instruction set, and compiler
  - SPEC (System Performance Evaluation Corporation) benchmarks, LINPACK, Winstone, gaming suites, etc
- Processor time, $T$
  - Time processor is busy executing instructions
  - Depends on processing system (CPU and memory) and compiler
  - Metrics used include xFLOPS (x floating point operations per second), xIPS (x instructions per second); e.g. GFLOPS, MIPS
High Performance Computing

- What performance is considered high?
  - No fixed value. It has increased with time. Yesterday’s supercomputing performance is now considered normal desktop performance.

- Cost –
  - performance
    - High performance achieved at significant higher cost
    - Performance gains at the higher end are expensive

Why High Performance Computing?

- Need
  - Computationally intensive problems: computational fluid mechanics, finite element analysis, simulation of engineering systems
  - Large scale problems: weather forecasting, environmental impact simulation, DNA sequencing, data mining

- High performance computing (HPC) is driven by need. Historically, applied scientists and engineers have fueled research and development in HPC systems (hardware, software, compilers, languages, networking, etc).
  - Reducing execution and processor time, $T$
  - Reducing $T$ crucial for HPC
Basic Performance Equation

- Processor time, $T$
  \[ T = \frac{(N \times S)}{R} \]

  - $N =$ No. of instructions executed by processor
  - $S =$ Average no. of basic steps per instruction
  - $R =$ Clock rate

- How can performance be increased?
  - Buy a higher performance computer?
    - Wrong!
  - Decrease $N$ and $S$, increase $R$; how?
    - Buy a higher performance computer? Yes, but…

Enhancing Performance

- Increase concurrency and parallelism
  - Operation level parallelism: pipelining and superscalar operation
  - Task parallelism: Loops, threads, multi-processor execution

- Improve utilization of hardware features
  - Caches
  - Pipelines
  - Communication links (e.g. bus)

- Develop algorithms and software that take advantage of hardware and compiler features
  - Programming languages and compilers
  - Software libraries
  - Algorithm design
Programming Models/Techniques

- Single processor performance issues
  - Data locality (cache and register tuning)
  - Data dependency (pipelining and superscalar ops)
  - Fine-grained parallelism (pipelining, threads, etc)

- Parallel programming models
  - Distributed memory or message passing programming
    - MPI (message passing interface)
  - Shared memory or global address space programming
    - POSIX threads (Pthreads)
    - OpenMP

---

Operation Level Parallelism

```c
for (i = 0; i < nrepeats; i++)
    w = 0.999999*w + 0.000001; /* O2 */

for (i = 0; i < nrepeats; i++) {
    w = 0.999999*w + 0.000001;  /* O4 */
    x = 0.999999*x + 0.000001;
}

for (i = 0; i < nrepeats; i++) {
    w = 0.999999*w + 0.000001; /* O8 */
    x = 0.999999*x + 0.000001;
    y = 0.999999*y + 0.0000001;
    z = 0.999999*z + 0.0000001;
```

<table>
<thead>
<tr>
<th>Performance in MFLOPS</th>
<th>O2</th>
<th>O4</th>
<th>O8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster node (P4 1.6 GHz)</td>
<td>100</td>
<td>175</td>
<td>385</td>
</tr>
<tr>
<td>Cluster node (P3 800 MHz)</td>
<td>123</td>
<td>245</td>
<td>326</td>
</tr>
<tr>
<td>suraj (UltraSparc II-250)</td>
<td>35</td>
<td>50</td>
<td>64</td>
</tr>
</tbody>
</table>
Cache Effect on Performance (1)

- Simple matrix transpose

```c
int n;
float a[n][n], b[n][n];

for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++)
        a[i][j] = b[j][i];
}
```

Performance in million moves per second

<table>
<thead>
<tr>
<th>Machine</th>
<th>32 x 32</th>
<th>1000 x 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster node (P4)</td>
<td>40</td>
<td>22</td>
</tr>
<tr>
<td>Cluster node (P3)</td>
<td>36</td>
<td>23</td>
</tr>
<tr>
<td>suraj</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

Cache Effect on Performance (2)

- Matrix-vector multiplication

```c
/* Dot-product form of matrix-vector multiply */
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++)
        y[i] = y[i] + a[i][j]*x[j];
}
```

```c
/* SAXPY form of matrix-vector multiply */
for (j = 0; j < n; j++) {
    for (i = 0; i < n; i++)
        y[i] = y[i] + a[i][j]*x[j];
}
```

Performance in million moves per second

<table>
<thead>
<tr>
<th>Machine</th>
<th>Dot-product</th>
<th>SAXPY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster node (P4)</td>
<td>2.1</td>
<td>1.9</td>
</tr>
<tr>
<td>Cluster node (P3)</td>
<td>9.6</td>
<td>8.8</td>
</tr>
<tr>
<td>suraj</td>
<td>9.6</td>
<td>4.0</td>
</tr>
</tbody>
</table>
Lessons

- The performance of a simple program can be a complex function of hardware and compiler
- Slight changes in the hardware or program can change performance significantly
- Since we want to write higher performing programs we must take into account the hardware and compiler in our programs even on single processor machines
- Since the actual performance is so complicated we need simple models to help us design efficient algorithms

Summary

- In theory, compilers can optimize our code for the architecture. In reality, compilers are still not that smart.
- Typical HPC applications are numerically intensive with major time spend in loops.
- To achieve higher performance, programmers need to understand and utilize hardware and software features to extract added performance.