Performance Analysis: Theory and Practice

Chris Mueller
Tech Tuesday Talk
July 10, 2007
Computational Science

- Traditional supercomputing
  - Simulation
  - Modeling
  - Scientific Visualization

- Key: problem (simulation) size
Informatics

- Traditional business applications, life sciences, security, the Web
- Workloads
  - Data mining
  - Databases
  - Statistical analysis and modeling
  - Graph problems
  - Information visualization
- Key: data size
Theory
Big-O Notation

Big-O notation is a method for describing the time or space requirements of an algorithm.

\[ f(x) = O(g(x)) \]

- \( f \) is an algorithm whose memory or runtime is a function of \( x \)
- \( O(g(x)) \) is an idealized class of functions of \( x \) that captures the general behavior of \( f(x) \)
- Only the dominant term in \( g(x) \) is used, constants are dropped

Example:

\[ f(x) = O(4x^2 + 2x + 1) = O(x^2) \quad f(x) \text{ is of order } n^2 \]
Asymptotic Analysis

Asymptotic analysis studies how algorithms scale and typically measures computation or memory use.

<table>
<thead>
<tr>
<th>Order</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>Constant</td>
</tr>
<tr>
<td></td>
<td>Table lookup</td>
</tr>
<tr>
<td>$O(\log n)$</td>
<td>Logarithmic</td>
</tr>
<tr>
<td></td>
<td>Binary search</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>Linear</td>
</tr>
<tr>
<td></td>
<td>Dot product</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>Quadratic</td>
</tr>
<tr>
<td></td>
<td>Comparison matrix</td>
</tr>
<tr>
<td>$O(a^n)$</td>
<td>Exponential</td>
</tr>
<tr>
<td></td>
<td>Multiple sequence alignment</td>
</tr>
</tbody>
</table>

\[
y = \text{table}[\text{idx}]
\]

\[
\text{for } x, y \text{ in } \text{zip}(X, Y):
\]
\[
\text{result } += x \times y
\]
Example: Dotplot

```
1. function dotplot(qseq, sseq, win, strig):
2.
3.     for q in qseq:
4.         for s in sseq:
5.             if CompareWindow(qseq[q:q+win], s[s:s+win], strig):
6.                 AddDot(q, s)

win = 3, strig = 2
```
Dotplot Analysis

*Dotplot is a polynomial time algorithm that is quadratic when the sequences are the same length.*

\[ O(m \times n \times \text{win} \times \text{comp}) \approx O(nm) \text{ or } O(n^2) \text{ if } m = n \]

- m is the number of elements read from q
- n is the number of elements read from s
- win is the number of comparisons for a single dot
- comp is the cost of a single character comparison
- win and comp are much smaller than the sequence lengths

```python
1. def dotplot(qseq, sseq, win, strig):
2.     for q in qseq:
3.         for s in sseq:
4.             if CompareWindow(qseq[q:q+win],
5.                             s[s:s+win], strig):
6.                 AddDot(q, s)
```
Unfortunately, real computers are not idealized compute engines with fast, random access to infinite memory stores…

Clock speed: 3.5 GHz
Data access occurs over a memory bus that is typically much slower than the processor…

Clock speed: 3.5 GHz
Bus speed: 1.0 GHz
Memory Hierarchy

Data is stored in a cache to decrease access times for commonly used items…

Clock speed: 3.5 GHz
Bus speed: 1.0 GHz
L1 cache access: 6 cycles
Memory Hierarchy

Caches are also cached…

Clock speed: 3.5 GHz
Bus speed: 1.0 GHz
L1 cache access: 6 cycles
L2 cache miss: 350 cycles
Memory Hierarchy

For big problems, multiple computers must communicate over a network…

Clock speed: 3.5 GHz
Bus speed: 1.0 GHz
L1 cache access: 6 cycles
L2 cache miss: 350 cycles
Network: Gigabit Ethernet
Asymptotic Analysis, Revisted

The effects of the memory hierarchy lead to an alternative method of performance characterization.

\[ f(x) = \frac{\text{number of reads} + \text{number of writes}}{\text{number of operations}} \]

- \( f(x) \) is the ratio of memory operations to compute operations
- If \( f(x) \) is small, the algorithm is compute bound and can achieve good performance
- If \( f(x) \) is large, the algorithm is memory bound and limited by the memory system
Dotplot Analysis, Memory Version

The number of communication to computation ratio is based on the sequence lengths, size of the DNA alphabet, and number of matches:

\[ \frac{|\Sigma_{\text{DNA}}| n + m + \alpha nm}{nm} \]

\( \alpha \) is the percentage of dots that are recorded as matches and is the asymptotic limit of the algorithm.

1. `def dotplot(qseq, sseq, win, strig):
2.     for q in qseq:
3.         for s in sseq:
4.             if CompareWindow(qseq[q:q+win],
5.                     s[s:s+win], strig):
6.                 AddDot(q, s) \]
Parallel Scaling

- What can we do with our time?
- **Strong**
  - Fix problem size, vary \( N \) procs, measure speedup
  - Goal: Faster execution
- **Weak**
  - Vary problem size, vary \( N \) procs, fix execution time
  - Goal: Higher resolution, more data
Dotplot: Parallel Decomposition

Dotplot is naturally parallel and is easily decomposed into blocks for parallel processing.

Each block can be assigned to a different processor.

Overlap prevents gaps by fully computing each possible window.

Scaling is strong until the work unit approaches the window size.
Practice
Application Architecture

Software architecture and implementation decisions have a large impact on the overall performance of an application, regardless of the complexity.

Theory matters:
- Algorithm selection (linear search vs. binary search)
- Data structures (arrays vs. linked lists)

Experience matters:
- Abstraction penalties (functions, objects, templates… oh my!)
- Data flow (blocking, comp/comm overlap, references/copies)
- Language (Scripting vs. compiled vs. mixed)
Abstraction Penalties

Well-designed abstractions lead to more maintainable code. But, compilers can’t always optimize English.

Abstraction

Matrix class:
1. A = Matrix();
2. B = Matrix();
3. C = Matrix();
4. // Initialize matrices...
5. C = A + B * 2.0;

Penalty

Transform operation expands to two steps and adds a temporary Matrix instance:
1. Matrix tmp = B.operator*(2.0);
2. C = A.operator+(tmp);

Arrays of double:
1. A = double[SIZE];
2. B = double[SIZE];
3. C = double[SIZE];
4. // Initialize matrices...
5. for(i = 0; i < SIZE; ++i)

Transform loop is unrolled and the operation uses the fused multiply add instruction to run at near peak performance.
1. ...
2. fmadd c, a, b, TWO
3. ...

Programmers must code to the compiler to achieve high performance.
Algorithms can be implemented to take advantage of a processor’s memory hierarchy.

Naïve Matrix-Matrix Multiplication...

```
M
  C = A x B
```

```python
1. def gemm(A, B, C):
2.     for i in range(M):
3.         for j in range(N):
4.             for k in range(K):
```

...Optimized for PowerPC 970

```
M
  C = A x B
```

```python
1. def gemm(A, B, C, mc, kc, nc, mr, nr):
2.     for kk in range(0, K, kc):
3.         tA[:,kk:kk+nc] = A[:,kk:kk+nc]
4.     for jj in range(0, N, nc):
6.     for ii in range(0, M, mr):
7.         for jj in range(0, nc, nr):
8.             for kk in range(0, kc):
9.                 for ai in range(mr):
10.                a[ai].load(tA, ai * A_row_stride)
11.            for bj in range(nr):
12.                b[bj].load(tB, bj * B_col_stride)
13.            for ci in range(mr):
14.                for cj in range(nr):
15.                    c[ci][cj].v = fmad(a[ci], b[cj], c[ci][cj])
16.                store(c, C_aux[:,jj:jj+nc])
17.                C[jj:jj+nc] += C_aux
18.         return
```
Any language can be used as a high-performance language.* But, it takes a skilled practitioner to properly apply a language to a particular problem.

Python-based matrix-matrix multiplication versus hand coded assembly.

*(ok, any language plus a few carefully designed native libraries)
Dotplot: Application Architecture

- Algorithm
  - Pre-computed score vectors
  - Running window

- Data structures
  - Indexed arrays for sequences
  - `std::vector` for dots
    - `(row, col, score)`

- Abstractions
  - Inline function for saving results

- Data flow
  - Blocked for parallelism

- Language
  - C++
System Architecture

Features of the execution environment have a large impact on application performance.

- Processor
  - Execution Units
  - Scalar, Superscalar
  - Instruction order
  - Cache
  - Multi-core
  - Commodity vs exotic

- Node
  - Memory bandwidth

- Network
  - Speed
  - Bandwidth
Porting for Performance

- Recompile (cheap)
  - Limited by the capabilities of the language and compiler

- Refactor (expensive)
  - Can the system architecture fundamentally change the performance characteristics? If yes...
  - Targeted optimizations (“accelerator”)
    - Algorithm (SIMD, etc)
    - Data structures
    - System resources (e.g., malloc vs. mmap, GPU vs. CPU)
  - Backhoe
    - Complete redesign (high cost, potential for big improvement)
Dotplot: System-specific Optimizations

- SIMD instructions allowed the algorithm to process 16 streams in parallel.
- A memory mapped file replaced `std::vector` for saving the results.
- All input and output files were stored on the local system. Results were aggregated as a post-processing step.
- Manually in-lined functions
Dotplot: Results

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>SIMD 1</th>
<th>SIMD 2</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal</td>
<td>140</td>
<td>1163</td>
<td>1163</td>
<td>2193</td>
</tr>
<tr>
<td>NFS</td>
<td>88</td>
<td>370</td>
<td>400</td>
<td>-</td>
</tr>
<tr>
<td>NFS Touch</td>
<td>88</td>
<td>-</td>
<td>446</td>
<td>891</td>
</tr>
<tr>
<td>Local</td>
<td>-</td>
<td>500</td>
<td>731</td>
<td>-</td>
</tr>
<tr>
<td>Local Touch</td>
<td>90</td>
<td>-</td>
<td>881</td>
<td>1868</td>
</tr>
</tbody>
</table>

The real application clock for dotplot is the memory bus.

- Base is a direct port of the DOTTER algorithm
- SIMD 1 is the SIMD algorithm using a sparse matrix data structure based on STL vectors
- SIMD 2 is the SIMD algorithm using a binary format and memory mapped output files
- Thread is the SIMD 2 algorithm on 2 Processors
Application “Clock”

Application performance can be limited by a number of different factors. Identifying the proper clock is essential for efficient use of development resources.

- System clock (GHz)
  - Compute bound
  - Example: Matrix-matrix multiplication
- Memory bus (MHz)
  - Memory bound
  - Example: Data clustering, GPU rendering, Dotplot
- Network (Kbs-Mbs)
  - Network bound
  - Example: Parallel simulations, Web applications, Grid
- User (FPS)
  - User bound
  - Example: Data entry, small-scale visualization
Conclusions

- Theory is simple and can be useful

- Practice is messy
  - Practice may not agree with theory...
  - Many factors and decisions affect final performance
    - Application architecture
    - System implementation
    - Programmer skill
    - Programmer time
Thank You!

Questions?