Introduction

CIS607 Seminar on Performance Modeling and Optimization
Spring 2014
Boyana Norris
Logistics

- Course wiki: [http://ix.cs.uoregon.edu/~norris/seminar](http://ix.cs.uoregon.edu/~norris/seminar)
- Seminar meets on Mondays, 2-3:20pm in 200 DES
- Office hours: whenever my door is open and by appointment
- Communications
  - Piazza
  - Google drive folder
  - Email

- First order of business: pick a week to lead discussion
Topics of this seminar

- Performance modeling (and some simulation)
- Performance optimization
Performance modeling

- What is performance?
  - Time, system throughput, responsiveness, power efficiency, ...

- What is performance modeling?
  - A model can be constructed to represent some aspect of the dynamic behaviour of a system.

- Why consider it?
  - Identify bottlenecks, other performance issues
  - Inform optimization
  - Predict performance on different systems, at different scales, etc...

- Approaches to modeling
  - Analytical
  - Empirical
    - Profile and trace-based
    - Simulation
  - Combinations of the above
Simple example: dense matrix-matrix product

- Consider square matrix multiplication

```c
for (i = 0; i < n; ++i) {
    for (j = 0; j < n; ++j) {
        C[i][j] = 0;
        for (k = 0; k < n; ++k)
            C[i][j] += A[i][k]*B[k][j];
    }
}
```

![Diagram of matrix multiplication](image)
Example (cont.)

Simple model:
- Consider two types of memory (fast and slow):
  - $m =$ words read from slow memory
  - $t_m =$ slow memory operation time
  - $f =$ number of FLOPS (floating-point operations)
  - $t_f =$ time per FLOP
  - $q = f/m =$ average FLOPS / slow memory access

Time:

$$f t_f + m t_m = f t_f (1 + (t_m / t_f)) / q$$

Important ratios:
- $t_m / t_f =$ machine balance (smaller is better)
- $q =$ computational intensity (larger is better)
Model details

What is the complexity of these operations?
1. Dot product \[ a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \quad ; \quad m=??, f=?? \]
2. Matrix-vector product: \[ m=??, f=?? \]
3. Matrix-matrix product: \[ m=??, f=?? \]

(These are examples of BLAS level 1, 2, and 3 routines that are principal building blocks of most numeric computations.)
Model details

What is the complexity of these operations? What is the computational intensity $q$?

1. Dot product $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$:
   
   $m=2n$ words, $f=2n$ FLOPS, $q = f/m \sim= 1$

2. Matrix-vector product:
   
   $m=n^2+3n$ words, $f=2n^2$ FLOPS, $q \sim= 2$
3. Matrix-matrix product:

Number of slow memory references on unblocked matrix multiply

\[ m = n^3 \text{ read each column of } B \text{ } n \text{ times} \]

\[ + n^2 \text{ read each row of } A \text{ once} \]

\[ + 2n^2 \text{ read and write each element of } C \text{ once} \]

\[ = n^3 + 3n^2 \]

So \( q = \frac{f}{m} = \frac{2n^3}{(n^3 + 3n^2)} \)

\[ \approx 2 \text{ for large } n, \text{ no improvement over matrix-vector multiply} \]
What about parallel algorithms?

Parallel matrix-matrix multiplication: things get hairy, fast

http://www.mcs.anl.gov/~itf/dbpp/text/node45.html
Performance optimization

Suppose we understand the performance issues of our code with the help of performance analysis and modeling. What is next?

Decide on an optimization approach:
- Leave it to the compiler (most common choice)
- Consider a different algorithm (best first choice!)
- Consider a different, already optimized implementation
- Manually optimize the low-level implementation
- Use a source-to-source tool to automatically optimize (portions of) the implementation
Example: Blocked (tiled) matrix-matrix product

Consider A,B,C to be N by N matrices of b by b subblocks where b=n / N is called the block size

```java
for (i = 0; i < n; ++i) {
    for (j = 0; j < n; ++j) {
        {read block C[i][j] into fast memory}
        for (k = 0; k < n; ++k) {
            {read block A[i][k] into fast memory}
            {read block B[k][j] into fast memory}
            C[i][j] += A[i][k]*B[k][j]; // multiply blocks
        }
        {write block C(i,j) back to slow memory}
    }
}
```

Acknowledgment: based on slides by Kathy Yellick UCB and Larry Carter UCSD
Blocked (tiled) matrix-matrix product

N = n/b = 4

\[ C[0][0] = C[0][0] + A[0][0] \times B[0][0] \]
Blocked (tiled) matrix-matrix product

\[ N = \frac{n}{b} = 4 \]

\[
\begin{align*}
C[0][0] &= C[0][0] + A[0][1] \times B[1][0] \\
\end{align*}
\]
Blocked (tiled) matrix-matrix product

\[ N = \frac{n}{b} = 4 \]

\[ \begin{array}{ccc}
C[0][0] & = & C[0][0] + A[0][2] \times B[2][0] \\
\end{array} \]
Blocked (tiled) matrix-matrix product

\[ N = \frac{n}{b} = 4 \]

\[
\begin{array}{c}
C[0][0] = C[0][0] + A[0][3] * B[3][0]
\end{array}
\]
Blocked (tiled) matrix-matrix product

\[ \text{N} = \frac{n}{b} = 4 \]

\[ \begin{array}{ccc}
C[0][1] & = & C[0][1] + A[0][0] \times B[0][1] \\
\end{array} \]
Blocked (tiled) matrix-matrix product

\[ N = \frac{n}{b} = 4 \]

\[
\begin{array}{c}
\text{C}[0][1] \\
\end{array} = \begin{array}{c}
\text{C}[0][1] \\
\end{array} + \begin{array}{c}
\text{A}[0][1] \\
\end{array} \times \begin{array}{c}
\text{B}[1][1] \\
\end{array}
\]

Acknowledgment: based on slides by Kathy Yellick UCB and Larry Carter UCSD
Blocked (tiled) matrix-matrix product

\[ N = n/b = 4 \]

\[
\begin{align*}
C[0][1] &= C[0][1] + A[0][2] \times B[2][1] \\
\end{align*}
\]
Blocked (tiled) matrix-matrix product

\[ N = n/b = 4 \]

\[
\begin{align*}
\begin{bmatrix}
C[0][1]
\end{bmatrix}
&= \\
\begin{bmatrix}
C[0][1]
\end{bmatrix} + \\
\begin{bmatrix}
A[0][3]
\end{bmatrix} \times \\
\begin{bmatrix}
B[3][1]
\end{bmatrix}
\end{align*}
\]
Blocked (tiled) matrix-matrix product

Recall:
- $m$ is the memory traffic between slow and fast memory (words)
- each matrix has $n \times n$ elements and $N \times N$ blocks, each of size $b \times b$
- $f$ is the number of floating-point ops, $2n^3$ for this problem
- $q = f/m$ is our measure of algorithm efficiency in the memory system
- Then:

$$m = N^3 n^2 \quad \text{read each block of } \mathbf{B} \ N^3 \text{ times } (N^3 \times n/N \times n/N)$$
$$+ N^3 n^2 \quad \text{read each block of } \mathbf{A} \ N^3 \text{ times}$$
$$+ 2n^2 \quad \text{read and write each block of } \mathbf{C} \ \text{once}$$
$$= (2N + 2) \times n^2$$

So computational intensity $q = f/m = 2n^3 / ((2N + 2) \times n^2)$

$$\sim n / N = b \ \text{for large } n$$

So we can improve performance by increasing the blocksize $b$
Can be much faster than matrix-vector multiply ($q=2$)

Acknowledgment: based on slides by Kathy Yellick UCB and Larry Carter UCSD
Using models to understand machines

The blocked algorithm has computational intensity \( q \approx b \)

- The larger the block size, the more efficient the algorithm will be
- **Limit**: All three blocks from A, B, and C must fit in fast memory (cache), so we cannot make these blocks arbitrarily large
- Assume your fast memory has size \( M_{\text{fast}} \)

\[
3b^2 \leq M_{\text{fast}}, \text{ so } q \approx b \leq \sqrt{M_{\text{fast}}/3}
\]

- To build a machine to run matrix-matrix product at the peak arithmetic speed of the machine, we need a fast memory of size

\[
M_{\text{fast}} \geq 3b^2 \approx 3q^2 = 3(T_m/T_f)^2
\]

- This size is reasonable for L1 cache, but not for register sets.
- Note: analysis assumes that it possible to schedule instructions perfectly.

Acknowledgment: based on slides by Kathy Yellick UCB and Larry Carter UCSD
Late 90s / early 2000s situation

<table>
<thead>
<tr>
<th></th>
<th>t_m/t_f</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ultra 2i</td>
<td>24.8186</td>
<td>14.8</td>
</tr>
<tr>
<td>Ultra 3</td>
<td>14</td>
<td>4.7</td>
</tr>
<tr>
<td>Pentium 3</td>
<td>6.25</td>
<td>0.9</td>
</tr>
<tr>
<td>Pentium3M</td>
<td>10</td>
<td>2.4</td>
</tr>
<tr>
<td>Power3</td>
<td>8.75</td>
<td>1.8</td>
</tr>
<tr>
<td>Power4</td>
<td>15</td>
<td>5.4</td>
</tr>
<tr>
<td>Itanium1</td>
<td>36</td>
<td>31.1</td>
</tr>
<tr>
<td>Itanium2</td>
<td>5.5</td>
<td>0.7</td>
</tr>
</tbody>
</table>
More recent architectures (> 2010)

<table>
<thead>
<tr>
<th>Processor</th>
<th>$t_m / t_f$</th>
<th>required (KB)</th>
<th>actual L1 (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-socket quad-core AMD Opteron Family 10h “Barcelona”</td>
<td>67 FLOPS/Word</td>
<td>13,467</td>
<td>1,024 (64 per core)</td>
</tr>
<tr>
<td>2-socket 6-core Intel Xeon 5600 “Westmere”</td>
<td>31 FLOPS/Word</td>
<td>2,883</td>
<td>768 (64 per core)</td>
</tr>
<tr>
<td>2-socket 8-core Xeon E5 “Sandy Bridge”</td>
<td>35 FLOPS/Word</td>
<td>3,675</td>
<td>512 (32 data + 32 instr. per core)</td>
</tr>
<tr>
<td>Xeon Phi</td>
<td>~48 FLOPS/Word</td>
<td>6,912</td>
<td>1,952 (32 data + 32 instr. per core, 61 cores)</td>
</tr>
</tbody>
</table>

Not useful for multicore chips -- distributed L1, cores don't share!
Modeling vs Analysis

❖ Nice paper by Torsten Hoeffler:

❖ Analysis:
  ➢ understanding the performance of some implementation
    ■ for some specific use cases/workloads
    ■ for a specific architecture

❖ Modeling
  ➢ Also helps to understand performance
    ■ over all workloads
    ■ parameterized for different architectures
  ➢ Can also use models to
    ■ Automate/optimize application configurations (selecting algorithms, etc.)
    ■ Runtime adaptation
    ■ Code generation and autotuning (offline and dynamic)
  ➢ Always remember, models are approximate
Modeling approaches

- **Analytical (by human)**
  - Deterministic, e.g., closed form continuous functions
  - Statistical, e.g., Petri nets

- **Empirical (automated)**
  - Data fitting
  - Machine learning, e.g., decision trees, neural nets, SVM, ...
  - Simulation, e.g., DES, MP, PN (hybrid analytical/empirical)

- **Static analysis (automated)**
  - Source or object code analysis

- **A combination of the above**
Typical analytical modeling “workflow”

Analytic performance modeling of HPC applications is usually performed in three phases:
1. Identify the performance-critical input parameters (through analysis of empirical data or
2. Formulate and test a hypothesis about the performance as function of the performance-critical (input) parameters
3. Parametrize the function

Outcome: Human-readable mathematical representations of software performance metrics.
Emprical “workflow”

1. Run experiments sampling the workloads of interest
2. Collect detailed performance data
3. Generate models
   a. Data fitting -- decide on a simple (e.g., linear, log, quadratic) function and then use least-squares-based approaches to determine the parameters
   b. Machine learning (more sophisticated data fitting) -- different approaches have somewhat different procedures, but generally you construct the model based on the training data, then test against known values for inputs not used in training, then grow the training set if needed

Outcome: Non-human-readable models (in most cases); quality depends on the choice of functions (regression) or training set (ML)
Simulation “workflow”

1. Create a small benchmark
2. Run in simulator, measuring metric of interest
   a. Use the result as the function value for given input, or
   b. Generate data for creating a model by using regression or ML

Outcome: A single data point of a reduced representation of the code for each input, which may take 100x or more to generate than running the application (but can be done for architectures that don’t exist).
Optimizations

- Tiling alone not enough
  - must find "best" block size
- Other practical optimizations
  - Tiling for multiple levels of memory
    - registers, L1, L2, L3, TLB
  - Exploiting fine-grained parallelism in processor
    - pipelining, vectorization
  - Loop unrolling (exposes instruction-level parallelism)
  - Loop fusion (reuse)
  - Scalar replacement (reuse, fewer loads)
  - ...
- Compiler interactions!
- Runtime system(s) interactions!
- Automatic optimization (incl. autotuning) -- active research area
  - Compilers
  - ATLAS, PhiPAC, Orio, CHiLL, ...