Memory hierarchies
(2/2)

Prof. Rich Vuduc <richie@cc>
Feb. 20, 2009
Quick recap:
An algorithmic view of the memory hierarchy
Idea 1: Machine balance & computational intensity.

\[
\begin{align*}
\text{flops} &= f \sim n^3 \\
\text{mops} &= m \sim \frac{n^3}{b} \\
q &= \frac{f}{m} \sim b \\
\frac{T}{f \cdot \tau} &\sim 1 + \frac{\alpha}{\tau} \cdot \frac{1}{b}
\end{align*}
\]
Idea 2: I/O optimality.

Theorem [Hong & Kung (1981)]: Any schedule of conventional mat-mul must transfer $\Omega\left(\frac{n^3}{\sqrt{Z}}\right)$ words between slow and fast memory, where $Z < \frac{n^2}{6}$.

Last time, we did intuitive proof by Toledo (1999)

Historical note: Rutledge & Rubinstein (1951—52)

So cached block matrix multiply is asymptotically optimal.

\[ b = O\left(\sqrt{Z}\right) \implies m = O\left(\frac{n^3}{b}\right) = O\left(\frac{n^3}{\sqrt{Z}}\right) \]
Theory vs. Practice

Recall: 1-core peak

Naïve parallelization (4-threads, OpenMP): 1.5x
Large, Complex Tuning Spaces

K. Jiang + A. Salahuddin (CSE 6230, Fall 2008)
Simplifying assumptions

- Ignored flop/mop parallelism within processor → drop arithmetic term
- Assumed randomly-accessible fast memory
- Assumed no-cost fast memory access
- Memory cost is constant, charged per word
  - Ignored cache lines / block transfers
  - Ignored bandwidth
Goals for today

- What is relationship to data movement in a parallel system?
- Learn the basics of a “real” memory hierarchy
  - Caches, TLB, memory
  - Ideas extend to disk systems
- Putting it all together: Algorithm design and engineering
Quick follow-up on I/O optimality results

- Irony, Toledo, & Tiskin (J. PDC 2004):
  “Communication lower bounds for distributed-memory matrix multiplication”
Theorem: “Memory-communication trade-off”

- # of I/Os depends on per-processor memory, $M$
- Need $M \geq \frac{n^3}{2\sqrt{2} \cdot p \cdot \sqrt{M}} - M$

What does this mean for algorithm designers? for computer architects?
Theorem: “2-D lower bound”

- Assuming each processor has
  - \( M = \mu \cdot n^2 / p \)
  - \( p \geq 32\mu^3 \)
- Cannon, Fox, SUMMA algorithms all match in terms of volume of communication

\[
\text{per-processor I/Os} \geq \frac{n^2}{4\sqrt{2} \cdot \mu p}
\]
Bandwidth vs. latency

- Message cost model to send $n$ words

$$T(n) = \alpha + \frac{n}{\beta} \quad \alpha = \text{latency (time)}, \beta = \text{bandwidth (words/time)}$$

- I/O optimality results cover bandwidth term
  - For latency bound, divide our I/O results by max message size, $M$
  - Flurry of new research on latency-optimal algorithms: See Hoemmen and Demmel at UC Berkeley
Back to memory hierarchies:
What does a cache look like?
Fast

Slow

CPU

Registers

L1

L2

Main
Dealing with high memory latency

- Motivation for **caches & prefetching**
  - Data likely to be reused (**temporal locality**)
  - Likely to access adjacent data (**spatial locality**)
  - Bandwidth improves faster than latency

- **H/W** provides automated support
  - All loads cached automatically (LRU), and loaded in chunks (**cache line size**)
  - Typical to have a hardware prefetcher that detects simple patterns
Cache terms

- **“Hit”** = load/store data in cache, vs. **“Miss”**
- Cache data transferred in **blocks** or **lines**
- Conceptual categories of misses
  - **Compulsory**: First access
  - **Capacity**: Cache cannot contain all blocks needed to execute the program
  - **Conflict**: Block replaced but then accessed soon thereafter
Cache design questions

- **Placement**: Where in cache does a block go?
  - Associativity

- **Replacement**: Replace which block on a miss?
  - Random, LRU, FIFO (round-robin)

- **Write strategy**: What happens on a write?
  - Write-through vs. write-back
  - Write-allocate or not
Placement

Q: Where does block #12 go?

Memory

Cache

- Fully associative: Anywhere
- (2-way) set associative: \( \text{mod}(12, 4) = 0 \)
- Direct mapped \( \text{mod}(12, 8) = 4 \)
Replacement

- Set- or fully-associative
  - Random: “Easy” to implement, but non-determinism makes life complicated for us
  - Least recently used (LRU): Hard to implement efficiently as associativity increases
  - Round-robin (first-in, first-out): Often used when associativity is high
- Victim cache: Hold evicted blocks
- Inclusive vs. exclusive multi-level caches
Write policies

- **Cache hit**
  - Write-through: ↑ traffic, but simplifies coherence
  - Write-back: Write only on eviction

- **Cache miss**
  - No write allocate: Write to main memory only
  - Allocate: Fetch into cache
Cache terms (refresher)

- Motivating intuition: Spatial & temporal locality
  - Cache hit vs. miss
  - 3C’s: compulsory, capacity, conflict
- Cache design space
  - Line size
  - Associativity: fully, k-way, direct
  - Replacement: random, LRU, round-robin (FIFO)
  - Write policies: write-thru, write-back, allocate
  - Victim cache, inclusive vs. exclusive
But how does the M.H. affect me? **Saavedra-Barrera benchmark**: Strided-stream through array, measuring average access time.

for $L$ in 2, 4, 8, ... do  // For each array size, $L$
for $s$ in 2, 4, 8, ..., $L/2$ do  // For each stride, $s$
  // Start timer
  for many trials do
    for $k = 0$ to $L$ step $s$ do  // For each element, in steps of $s$
      read $X[k]$
  // Stop timer and report average read time per element
What do we expect?

Let $T = \text{avg. read time}$ (cycles).

Array size: $n$ words
Stride: $s$ words
Line size: $l$ words
Assoc.: Direct-mapped
Replacement: LRU
What do we expect?

Mem. latency: $\alpha$ cycles
Cache size: $Z$ words
Line size: $l$ words
Cache latency: 1 cycle
Assoc.: Direct-mapped
Replacement: LRU
Array size: $n$ words
Stride: $s$ words

Let $\tau = \text{avg. read time}$ (cycles).

$n \leq Z \implies \tau = ?$
What do we expect?

Mem. latency: $\alpha$ cycles
Cache size: $Z$ words
Line size: $l$ words
Cache latency: 1 cycle
Assoc.: Direct-mapped
Replacement: LRU
Array size: $n$ words
Stride: $s$ words

Let $\tau = \text{avg. read time}$ (cycles).

$n \leq Z \implies \tau = 1$

Array cache-resident, so $\tau$ independent of $s$. 
What do we expect?

Mem. latency: $\alpha$ cycles
Cache size: $Z$ words
Line size: $l$ words
Cache latency: 1 cycle
Assoc.: Direct-mapped
Replacement: LRU
Array size: $n$ words
Stride: $s$ words

Let $\tau = \text{avg. read time}$ (cycles).

$n \leq Z \implies \tau = 1$

$n > Z \implies \tau = ?$

Array cache-resident, so $\tau$ independent of $s$. 

Array Cache Node
What do we expect?

Mem. latency: $\alpha$ cycles
Cache size: $Z$ words
Line size: $l$ words
Cache latency: 1 cycle
Assoc.: Direct-mapped
Replacement: LRU
Array size: $n$ words
Stride: $s$ words

Let $\tau = \text{avg. read time}$ (cycles).

$n > Z$ and $s = 1$

$\Rightarrow \quad \tau = ?$
What do we expect?

Mem. latency: $\alpha$ cycles
Cache size: $Z$ words
Line size: $l$ words
Cache latency: 1 cycle
Assoc.: Direct-mapped
Replacement: LRU
Array size: $n$ words
Stride: $s$ words

Let $\tau = \text{avg. read time}$ (cycles).

$n > Z$ and $s = 1$

$\Rightarrow \quad \tau = \frac{\alpha + (l - 1)}{l}$

Full latency on 1st word only.
What do we expect?

Mem. latency: $\alpha$ cycles
Cache size: $Z$ words
Line size: $l$ words
Cache latency: 1 cycle
Assoc.: Direct-mapped
Replacement: LRU
Array size: $n$ words
Stride: $s$ words

Let $\tau = \text{avg. read time}$ (cycles).

$$n > Z \quad \text{and} \quad s \leq l$$

$\Rightarrow \quad \tau = ?$
What do we expect?

Mem. latency: $\alpha$ cycles
Cache size: $Z$ words
Line size: $l$ words
Cache latency: 1 cycle
Assoc.: Direct-mapped
Replacement: LRU
Array size: $n$ words
Stride: $s$ words

Let $\tau = \text{avg. read time}$ (cycles).

$n > Z$ and $s \leq l$

$\implies \tau = \frac{\alpha + \frac{l}{s} - 1}{\frac{l}{s}}$
What do we expect?

Mem. latency: $\alpha$ cycles
Cache size: $Z$ words
Line size: $l$ words
Cache latency: 1 cycle
Assoc.: Direct-mapped
Replacement: LRU
Array size: $n$ words
Stride: $s$ words

Let $\tau =$ avg. read time (cycles).

$n > Z$ and $s > l$

$\Rightarrow \quad \tau = ?$
What do we expect?

Mem. latency: $\alpha$ cycles
Cache size: $Z$ words
Line size: $l$ words
Cache latency: 1 cycle
Assoc.: Direct-mapped
Replacement: LRU
Array size: $n$ words
Stride: $s$ words

Let $\tau = \text{avg. read time}$ (cycles).

$n > Z$ and $s > l$  \[\implies \tau = \alpha\]

Full latency on 1st word only.
\[ n \leq Z \implies \tau = 1 \]
$n \leq Z \implies \tau = 1$
\[ n \leq Z \implies \tau = 1 \]

\[ n > Z \quad \text{and} \quad s > l \implies \tau = \alpha \]
\[ n \leq Z \implies \tau = 1 \]
\[ n > Z \quad \text{and} \quad s > l \implies \tau = \alpha \]
$n > Z$ and $s = 1$ 

$\implies \quad \tau = \frac{\alpha + (l - 1)}{l}$
\[ n > Z \quad \text{and} \quad s = 1 \]

\[ \Rightarrow \quad \tau = \frac{\alpha + (l - 1)}{l} \]
\[ n > Z \text{ and } s \leq l \]

\[ \implies \tau = \frac{\alpha + \frac{l}{s} - 1}{\frac{l}{s}} \]
\( n > Z \) and \( s \leq l \)

\[ \Rightarrow \quad \tau = \alpha + \frac{l}{s} - 1 \]
Average Memory Access Time (Saavedra-Barerra) — Sun Ultra IIi (333 MHz)
Average Memory Access Time (Saavedra-Barerra) — Sun Ultra IIi (333 MHz)

- **L1:** 16 KB, 16 B lines
- **L2:** 2 MB, 64 B lines
Average Memory Access Time (Saavedra-Barerra) — Sun Ultra IIli (333 MHz)

- L1: 16 KB, 16 B lines
- L2: 2 MB, 64 B lines
- TLB: 8 KB page, 32 entries

Graph showing the relationship between memory access time and stride (bytes) for different memory sizes.
Average Memory Access Time (Saavedra-Barerra) — Pentium III (Katmai; 550 MHz)

- L1: 16 KB, 32 B lines
- L2: 512 KB, 32 B lines
- TLB: 4 KB page, 64 entries
- L1: 16 KB, 32 B lines
- L2: 512 KB, 32 B lines
- TLB: 4 KB page, 64 entries
“TLB” is part of the memory hierarchy

- Virtual address space – Why?
- Translation Look-aside Buffer helps manage virtual memory
  - Divide address space into pages (4–32 KB typical)
  - Page table maps virtual to physical addr & whether page in mem or on disk
  - TLB caches translations
- May be set-associative or fully-associative
- Conceptually like a cache with large block size, i.e., 1 page
  - May have multiple levels of TLB, just like cache
  - Can prefetch to hide cache misses, but not TLB misses
Fast

Slow

Main

Registers

L1

TLB

L2
Bridging the gap between theory & practice

Sources:

Sources

- Course on tuning by Clint Whaley (UTSA)
- Course on tuning by Markus Püschel
Matrix Dimension

Mflop/s

1 core on 4–core AMD Opteron @ 2.0 GHz

Max(Blocked)

Naive
Some questions

- Block size selection?
- Why the dips?
- For which level(s) of memory hierarchy do we optimize explicitly?
Block size constraints

- Assume:
  - Assume 1 FMA / cycle \Rightarrow \text{ideal time} = n^3 \text{ cycles}
  - Square \( b \times b \) blocks
  - Two-level memory (slow & fast)
- Recall capacity constraint: Lower bound

\[ b \leq \sqrt{\frac{Z}{3}} \]
Upper-bound: Bandwidth constraint

\[ f \equiv \text{flops} = 2n^3 \]
\[ m \equiv \text{mops} \approx \frac{2n^3}{b} \]
\[ \beta \equiv \text{Peak bandwidth (words/cycle)} \]
\[ \rho \equiv \text{Peak flop/cycle} \]
\[ \frac{m}{\beta} \leq \frac{f}{\rho} \implies b \geq \frac{\rho}{\beta} \]
\[ \downarrow \]
\[ \frac{\rho}{\beta} \leq b \leq \sqrt{\frac{Z}{3}} \]
Block size constraint

\[
\frac{\rho}{\beta} \leq b \leq \sqrt{\frac{Z}{3}}
\]

⇓

\[
\rho \sim 4 \frac{\text{flop}}{\text{cycle}}
\]

\[
\beta \sim 0.4 \frac{\text{double-words}}{\text{cycle}} \quad \text{(6.4 GB/s at 2 GHz)}
\]

\[
Z \sim 256 \text{ K-dwords (2 MB cache)}
\]

⇓

\[
10 \leq b \lesssim 295
\]
Multi-level blocking

- Idea: Block at multiple levels
  - Each cache, TLB, and CPU registers
  - Match multi-level memory hierarchy

- Following discussion follows:
\[ C \leftarrow C + A \cdot B \]

- "Matrix-matrix"
- "Matrix-panel"
- "Panel-matrix"
- "Panel-Panel" or "Fat Outer Product"
\[ C \leftarrow C + A \cdot B \]

“Matrix-matrix”

"Block-Panel"

"Panel-block"

“Fat Dot Product"
// Let $I, J, K = \text{blocks of indices}$

for $K \leftarrow \text{blocks 1 to } \frac{k}{b_k}$ do

for $I \leftarrow \text{blocks 1 to } \frac{m}{b_m}$ do

for $J \leftarrow \text{blocks 1 to } \frac{n}{b_n}$ do

$C_{IJ} \leftarrow C_{IJ} + A_{IK} \times B_{KJ}$
// “Block-panel” multiply
// Load $b_m \times b_k$ block of A into cache

for $J \leftarrow$ blocks 1 to $\frac{n}{b_n}$ do

// Load $b_k \times b_n$ block of B into cache
// Load $b_m \times b_n$ block of C into cache
$C_J \leftarrow C_J + A \times B_J$

// Store $b_m \times b_n$ block of C to memory

---

**Assumes:**

1. A, $B_J$, $C_J$ fit in cache (e.g., size $Z$)
2. Above $\Rightarrow$ Product runs at peak
3. A not evicted prematurely

$$b_m b_k + (b_k + b_m) b_n \leq Z$$
// “Block-panel” multiply
// Load $b_m \times b_k$ block of $A$ into cache
for $J \leftarrow$ blocks 1 to $\frac{n}{b_n}$ do
  // Load $b_k \times b_n$ block of $B$ into cache
  // Load $b_m \times b_n$ block of $C$ into cache
  $C_J \leftarrow C_J + A \times B_J$
  // Store $b_m \times b_n$ block of $C$ to memory

Assumes:
1. $A, B_J, C_J$ fit in cache (e.g., size $Z$)
2. Above $\Rightarrow$ Product runs at peak
3. $A$ not evicted prematurely

$$b_m b_k + (b_k + b_m)b_n \leq Z$$

$$f = 2b_m b_k n$$

$$m = b_m b_k + (b_k + 2b_m) n$$

$$q = \frac{2}{\frac{1}{n} + \left(\frac{1}{b_m} + \frac{2}{b_k}\right)}$$
Given a multi-level memory hierarchy, in what cache should “A” block live?

- Want large A block
- L1 cache usually quite small
- What about L2?

Assumes:
1. A, B_j, C_j fit in cache (e.g., size Z)
2. Above ⇒ Product runs at peak
3. A not evicted prematurely

\[
b_m b_k + (b_k + b_m) b_n \leq Z
\]

\[
\frac{2b_m b_k b_n}{\rho_1} \geq \frac{b_m b_k}{\beta_2}
\]

\[
\Downarrow
\]

\[
b_n \geq \frac{\rho_1}{2\beta_2}
\]

Typically, need \( b_n \geq 2 \).
**Assumes:**
1. $A, B_j, C_j$ fit in cache (e.g., size $Z$)
2. Above $\Rightarrow$ Product runs at peak
3. $A$ not evicted prematurely

\[
b_m b_k + (b_k + b_m) b_n \leq Z
\]
\[
b_n \geq \frac{\rho_1}{2\beta_2}
\]
What about the TLB?

Assumes:
1. \( A, B_j, C_j \) fit in cache (e.g., size \( Z \))
2. Above \( \Rightarrow \) Product runs at peak
3. \( A \) not evicted prematurely

\[
\begin{align*}
    b_m b_k + (b_k + b_m) b_n & \leq Z \\
    b_n & \geq \frac{\rho_1}{2\beta_2}
\end{align*}
\]
Considerations for TLB

- **Matrix**
  - n = 1024
  - Column-major order

- **TLB**
  - Page = 4 KB
  - 32 entries
What about the TLB?

Block of $A$ straddles pages, so re-pack on-the-fly
⇒ “Copy optimization”

Copy $B$ panel as well

Assumes:
1. $A, B_j, C_j$ fit in cache (e.g., size $Z$)
2. Above ⇒ Product runs at peak
3. $A$ not evicted early
4. Operands “fit in” TLB

\[
b_m b_k + (b_k + b_m) b_n \leq Z \\
\]
\[
b_n \geq \frac{\rho_1}{2^\beta_2}
\]
Panel-Block

Fat-Dot
// Let $I, J, K = \text{blocks of indices}$

\[
\text{for } K \leftarrow \text{blocks 1 to } \frac{k}{b_k} \text{ do}
\]

\[
\tilde{B} \leftarrow B_{K,*}
\]

\[
\text{for } I \leftarrow \text{blocks 1 to } \frac{m}{b_m} \text{ do}
\]

\[
\tilde{A} \leftarrow A_{IK}
\]

\[
\text{for } J \leftarrow \text{blocks 1 to } \frac{n}{b_n} \text{ do}
\]

\[
\tilde{C} \leftarrow \tilde{A} \times \tilde{B}_J \quad // \text{Compute in buffer, } \tilde{C}
\]

\[
C_{IJ} \leftarrow C_{IJ} + \tilde{C} \quad // \text{Unpack } \tilde{C}
\]
Which is better?
Dense Matrix Multiply Performance (Square $n \times n$ Operands) [333 MHz Sun Ultra 2i]

- Vendor
- Reg/insn–level + cache tiling + copy opt.
- Cache tiling + copy opt.
- Reference

Source: Vuduc, Demmel, Bilmes (IJHPCA 2004)
Dense Matrix Multiply Performance (Square $n \times n$ Operands) [800 MHz Intel Pentium III–mobile]

Source: Vuduc, Demmel, Bilmes (IJHPCA 2004)
Cache-oblivious matrix multiply (from Yotov, et al. SPAA ’07)
Memory model for analyzing cache-oblivious algorithms

- Two-level memory hierarchy
- \( Z = \) capacity of cache ("fast")
- \( L = \) cache line size
- Fully associative
- Optimal replacement
  - Evicts most distant use
  - Sleator & Tarjan (CACM 1985): LRU, FIFO w/in constant of optimal w/ cache larger by constant factor
- "Tall-cache:" \( Z \geq O(L^2) \)
  - Limits: See Brodal & Fagerberg (STOC 2003)
  - When might this not hold?
A recursive algorithm for matrix-multiply

- Divide all dimensions in half
- Bilardi, et al.: Use Gray code ordering

\[
\begin{align*}
\text{Cost (flops)} &= T(n) \\
&= \begin{cases} 
8 \cdot T\left(\frac{n}{2}\right) & n > 1 \\
O(1) & n = 1 
\end{cases} \\
&= O(n^3)
\end{align*}
\]
A recursive algorithm for matrix-multiply

- Divide all dimensions in half
- Bilardi, et al.: Use Gray-code ordering

I/O Complexity?
A recursive algorithm for matrix-multiply

- Divide all dimensions in half
- Bilardi, et al.: Use Gray-code ordering

No. of misses, with tall-cache assumption:

\[
Q(n) = \begin{cases} 
8 \cdot Q\left(\frac{n}{2}\right) & \text{if } n > \sqrt{\frac{Z}{3}} \\
3n^2 \frac{L}{L} & \text{otherwise}
\end{cases}
\leq \Theta \left(\frac{n^3}{L\sqrt{Z}}\right)
\]
Alternative: Divide longest dimension (Frigo, et al.)

Cache misses $Q(m, k, n) \leq \begin{cases} \Theta \left( \frac{mk+kn+mn}{L} \right) & \text{if } mk + kn + mn \leq \alpha Z \\ 2Q \left( \frac{m}{2}, k, n \right) & \text{if } m \geq k, n \\ 2Q \left( m, \frac{k}{2}, n \right) & \text{if } k > m, k \geq n \\ 2Q \left( m, k, \frac{n}{2} \right) & \text{otherwise} \end{cases}

= \Theta \left( \frac{mkn}{L\sqrt{Z}} \right)
Relax tall-cache assumption using suitable layout

Row-major

Need tall cache

Row-block-row

$M \geq \Omega(L)$

Morton Z

No assumption

Source: Yotov, et al. (SPAA 2007) and Frigo, et al. (FOCS ’99)
Outer Control Structure

Iterative, Recursive

Inner Control Structure

Statement, Recursive, Iterative

Mini-Kernel

Micro-Kernel

None / Compiler, Scalarized / Compiler, Belady / BRILA, Coloring / BRILA

ATLAS CGw/S, ATLAS Unleashed
Summary: Cache-oblivious engineering considerations

- Need to cut-off recursion
- Careful scheduling/tuning required at “leaves”
- Yotov, et al., report that full-recursion + tuned micro-kernel ≤ 2/3 best

Open issues

- Recursively-scheduled kernels worse than iteratively-schedule kernels — why?
- Prefetching needed, but how best to apply in recursive case?