Lecture 4: Memory Hierarchy and Cache

Cache: A safe place for hiding and storing things.
Webster's New World Dictionary (1976)

Terms

- flops - floating point operations (usually 64 bit)
  - + or * count as 1 flop
  - sqrt or / count as 4 flops or 2 flops
  - log counts as 8 flops
  - compare counts as 1 flop
- flops are different than cycles!
- flop/s - floating point operations per second
- Mflop/s - Million flop/s, Gflop/s, Tflop/s
- Theoretical peak performance

Execution Time

- Ambiguous, what is measured ? CPU time? Wallclock time?
- Wallclock time is preferred
  - With serial programs there will be some system overhead that will effect the time-to-completion.
  - In a parallel program the CPU time is totally inadequate because of sync and scheduling overhead on shared-memory systems and because of sync and communication overhead on distributed memory systems.

Tools for Performance Evaluation

- Timing and performance evaluation has been an art
  - Resolution of the clock
  - Issues about cache effects
  - Different systems
- Situation about to change
  - Today's processors have counters

Performance Counters

- Almost all high performance processors include hardware performance counters.
- On most platforms the APIs, if they exist, are not appropriate for a common user, functional or well documented.
- Existing performance counter APIs
  - Cray T3E
  - SGI MIPS R10000
  - IBM Power series
  - DEC Alpha pfm pseudo-device interface
  - Windows 95, NT and Linux

Performance Data (cont.)

- Cycle count
- Floating point instruction count
- Integer instruction count
- Instruction count
- Load/store count
- Branch taken / not taken count
- Branch mispredictions
- Pipeline stalls due to memory subsystem
- Pipeline stalls due to resource conflicts
- I/D cache misses for different levels
- Cache invalidations
- TLB misses
- TLB invalidations

PAPI Usage

- Application is instrumented with PAPI
- Will be layered over the best existing vendor-specific APIs for these platforms
- call papi_timer(t1, t2)
  - t1 is the process time
- call papi_flops(f1, f2)
  - f2 is the total # of floating point operations for the process

7

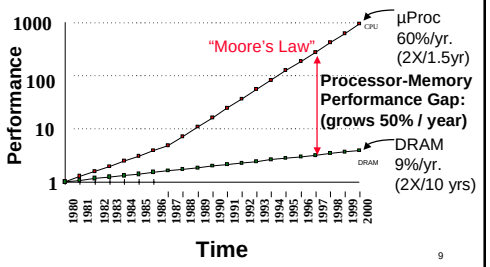
Cache and Its Importance in Performance

- **Motivation:**
  - Time to run code = clock cycles running code + clock cycles waiting for memory
  - For many years, CPU's have sped up an average of 50% per year over memory chip speed ups.
- **Hence, memory access is the bottleneck to computing fast.**
- **Definition of a cache:**
  - **Dictionary:** a safe place to hide or store things.
  - **Computer:** a level in a memory hierarchy.

8

Who Cares About the Memory Hierarchy?

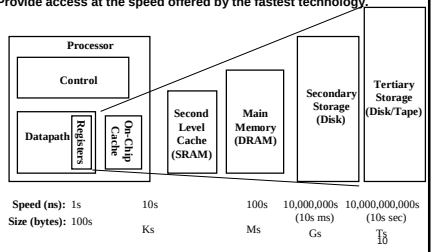
Processor-DRAM Memory Gap (latency)



9

A Modern Memory Hierarchy

- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.

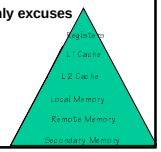


10

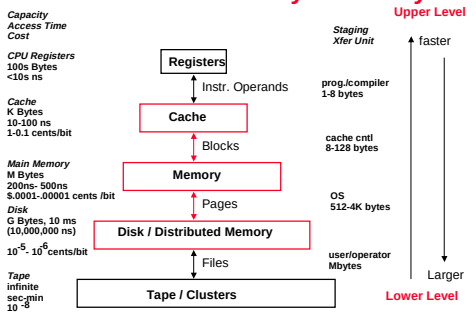
The Memory Hierarchy

1. Different memory subsystems have different speeds, sizes, and costs.
2. Smaller memory implies faster

   Slower memory implies cheaper.
3. The fastest memory is closest to the CPU while the slowest is further away.
4. Every memory level is a subset of any level further away.
5. Performance and cost savings are the only excuses for this mess.



Levels of the Memory Hierarchy



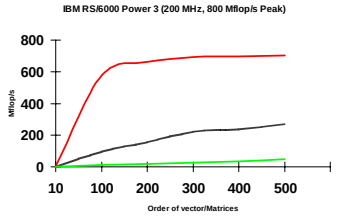
12

Uniprocessor Reality

- Modern processors use a variety of techniques for performance
  - caches
    - » small amount of fast memory where values are "cached" in hope of reusing recently used or nearby data
    - » different memory ops can have very different costs
  - parallelism
    - » superscalar processors have multiple "functional units" that can run in parallel
    - » different orders, instruction mixes have different costs
  - pipelining
    - » a form of parallelism, like an assembly line in a factory
- Why is this your problem?
  - » In theory, compilers understand all of this and can optimize your program; in practice they don't.

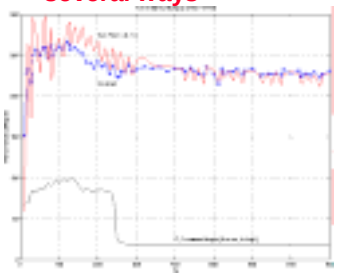
13

Matrix-multiply, optimized several ways



14

Matrix-multiply, optimized several ways



Speed of n-by-n matrix multiply on Sun Ultra-1/170, peak = 330 MFlops

15

Cache Basics

- Cache hit: a memory access that is found in the cache -- cheap
- Cache miss: a memory access that is not in the cache - expensive, because we need to get the data from elsewhere
- Consider a tiny cache (for illustration only)

X000	X001
X010	X011
X100	X101
X110	X111

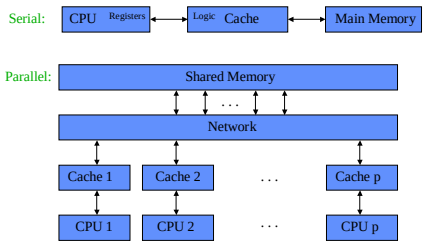
Address

tag	line	offset

- Cache line length: number of bytes loaded together in one entry
- Direct mapped: only one address (line) in a given range in cache
- Associative: 2 or more lines with different addresses exist

16

Diagrams



17

Tuning for Caches

1. Preserve locality.
2. Reduce cache thrashing.
3. Loop blocking when out of cache.
4. Software pipelining.

18

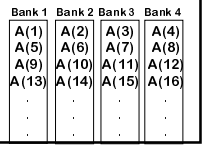
Registers

- Registers are the source and destination of most CPU data operations.
- They hold one element each.
- They are made of static RAM (SRAM), which is very expensive.
- The access time is usually 1-1.5 CPU clock cycles.
- Registers are at the top of the memory subsystem.

19

Memory Banking

- This started in the 1960's with both 2 and 4 way interleaved memory banks. Each bank can produce one unit of memory per bank cycle. Multiple reads and writes are possible in parallel.
  - Memory chips must internally recover from an access before it is reaccessed
- The bank cycle time is currently 4-8 times the CPU clock time and getting worse every year.
- Very fast memory (e.g., SRAM) is unaffordable in large quantities.
- This is not perfect. Consider a 4 way interleaved memory and a stride 4 algorithm. This is equivalent to non-interleaved memory systems.



The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 15 years, HW relied on localilty for speed

21

Principals of Locality

- Temporal: an item referenced now will be again soon.
- Spatial: an item referenced now causes neighbors to be referenced soon.
- Lines, not words, are moved between memory levels. Both principals are satisfied. There is an optimal line size based on the properties of the data bus and the memory subsystem designs.
- Cache lines are typically 32-128 bytes with 1024 being the longest currently.

22

Cache Sporting Terms

- Cache Hit: The CPU requests data that is already in the cache. We want to maximize this. The hit rate is the percentage of cache hits.
- Cache Miss: The CPU requests data that is not in cache. We want to minimize this. The miss time is how long it takes to get data, which can be variable and is highly architecture dependent.
- Two level caches are common. The L1 cache is on the CPU chip and the L2 cache is separate. The L1 misses are handled faster than the L2 misses in most designs.
- Upstream caches are closer to the CPU than downstream caches. A typical Alpha CPU has L1-L3 caches. Some MIPS CPU's do, too.

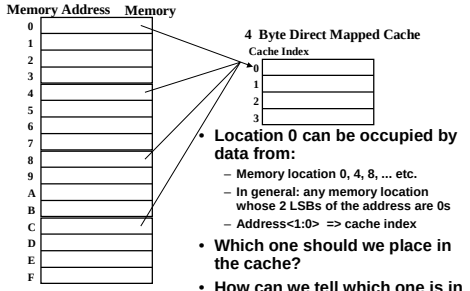
23

Unified versus Split Caches

- This refers to having a single or separate caches for data and machine instructions.
- Split is obviously superior. It reduces thrashing, which we will come to shortly..

24

Simplest Cache: Direct Mapped

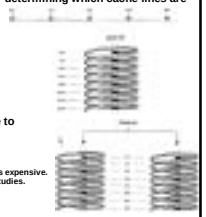


- Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> => cache index
- Which one should we place in the cache?
- How can we tell which one is in the cache?

25

Cache Mapping Strategies

- There are two common sets of methods in use for determining which cache lines are used to hold copies of memory lines.
- Direct: Cache address = memory address MODULO cache size.
- Set associative: There are N cache banks and memory is assigned to just one of the banks. There are three algorithmic choices for which line to replace:
  - Random: Choose any line using an analog random number generator. This is cheap and simple to make.
  - LRU (least recently used): Preserves temporal locality, but is expensive. This is not much better than random according to (biased) studies.
  - FIFO (first in, first out): Random is far superior.



26

What happens on a write?

- ***Write through***—The information is written to both the block in the cache and to the block in the lower-level memory.
- ***Write back***—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced in cache.
  - is block clean or dirty?
- Pros and Cons of each?
  - WT: read misses cannot result in writes
  - WB: no repeated writes to same location
- WT always combined with write buffers so that don't wait for lower level memory

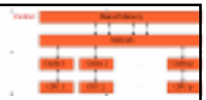
27

Cache Thrashing

- Thrashing occurs when frequently used cache lines replace each other. There are three primary causes for thrashing:
  - Instructions and data can conflict, particularly in unified caches.
  - Too many variables or too large of arrays are accessed that do not fit into cache.
  - Indirect addressing, e.g., sparse matrices.
- Machine architects can add sets to the associativity. Users can buy another vendor's machine. However, neither solution is realistic.

28

Cache Coherence for Multiprocessors



- All data must be coherent between memory levels. Multiple processors with separate caches must inform the other processors quickly about data modifications (by the cache line). Only hardware is fast enough to do this.
- Standard protocols on multiprocessors:
  - Snoopy: all processors monitor the memory bus.
  - Directory based: Cache lines maintain an extra 2 bits per processor to maintain clean/dirty status bits.
- False sharing occurs when two different shared variables are located in the in the same cache block, causing the block to be exchanged between the processors even though the processors are accessing different variables. Size of block (line) important.

29

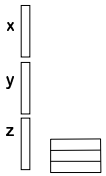
Processor Stall

- Processor stall is the condition where a cache miss occurs and the processor waits on the data.
- A better design allows any instruction in the instruction queue to execute that is ready. You see this in the design of some RISC CPU's, e.g., the RS6000 line.
- Memory subsystems with hardware data prefetch allow scheduling of data movement to cache.
- Software pipelining can be done when loops are unrolled. In this case, the data movement overlaps with computing, usually with reuse of the data.
- out of order execution, software pipelining, and prefetch.

30

Indirect Addressing

```
d = 0
do i = 1,n
  j = ind(i)
  d = d + sqrt( x(j)*x(j) + y(j)*y(j) + z(j)*z(j) )
end do
```



- **Change loop statement to**

```
d = d + sqrt( r(1,j)*r(1,j) + r(2,j)*r(2,j) + r(3,j)*r(3,j) )
```


- **Note that r(1,j)-r(3,j) are in contiguous memory and probably are in the same cache line (d is probably in a register and is irrelevant). The original form uses 3 cache lines at every instance of the loop and can cause cache thrashing.**

31

Cache Thrashing by Memory Allocation

```
parameter ( m = 1024*1024 )
real a(m), b(m)
```

- **For a 4 Mb direct mapped cache, a(i) and b(i) are always mapped to the same cache line. This is trivially avoided using padding.**

```
real a(m), extra(32), b(m)
```

- **extra is at least 128 bytes in length, which is longer than a cache line on all but one memory subsystem that is available today.**

32

Cache Blocking

- **We want blocks to fit into cache. On parallel computers we have p x cache so that data may fit into cache on p processors, but not one. This leads to superlinear speed up! Consider matrix-matrix multiply.**

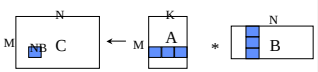
```
do k = 1,n
  do j = 1,n
    do i = 1,n
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
    end do
  end do
end do
```

- **An alternate form is ...**

33

Cache Blocking

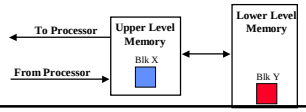
```
do kk = 1,n,nblk
  do jj = 1,n,nblk
    do ii = 1,n,nblk
      do k = kk,kk+nblk-1
        do j = jj,jj+nblk-1
          do i = ii,ii+nblk-1
            c(i,j) = c(i,j) + a(i,k) * b(k,j)
          end do
        . . .
end do
```



34

Memory Hierarchy: Terminology

- **Hit: data appears in some block in the upper level (example: Block X)**
  - **Hit Rate: the fraction of memory access found in the upper level**
  - **Hit Time: Time to access the upper level which consists of RAM access time + Time to determine hit/miss**
- **Miss: data needs to be retrieve from a block in the lower level (Block Y)**
  - **Miss Rate = 1 - (Hit Rate)**
  - **Miss Penalty: Time to replace a block in the upper level + Time to deliver the block the processor**
- **Hit Time << Miss Penalty (500 instructions on Alpha 21264!)**



35

Cache Measures

- ***Hit rate*: fraction found in that level**
  - **So high that usually talk about *Miss rate***
- **Average memory-access time**
  = Hit time + Miss rate x Miss penalty
  (ns or clocks)
- ***Miss penalty*: time to replace a block from lower level, including time to replace in CPU**
  - ***access time*: time to lower level**
    = f(latency to lower level)
  - ***transfer time*: time to transfer block**
    =f(BW between upper & lower levels)

36

‘‘Rules of Thumb’’

- 90/10 Locality Rule:
  - A program executes approximately 90% of its instructions in 10% of its code.
  - Implication: there is a ‘‘common case”.
- Size/Complexity versus Speed Rule:
  - In hardware and software, smaller (and/or less complex) is faster.
  - Implication: there is a way to make a common case ‘‘fast”

37

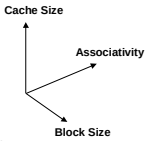
Why Smaller is Faster

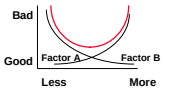
- Smaller means shorter path for physical signal propagation
  - ‘‘one foot is one nanosecond”
  - Smaller means less decoding time for logical interpretation
  - fewer items to choose between
  - shorter codes to distinguish between them
- Smaller means higher-grade technology can be employed
  - e.g., expensive ECL memory for fast cache, vector registers

38

Summary : The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation
- The optimal choice is a compromise
  - depends on access characteristics
    - » workload
    - » use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins





39

Lessons

- The actual performance of a simple program can be a complicated function of the architecture
- Slight changes in the architecture or program change the performance significantly
- Since we want to write fast programs, we must take the architecture into account, even on uniprocessors
- Since the actual performance is so complicated, we need simple models to help us design efficient algorithms
- We will illustrate with a common technique for improving cache performance, called blocking

40

Optimizing Matrix Addition for Caches

- Dimension A(n,n), B(n,n), C(n,n)
- A, B, C stored by column (as in Fortran)
- Algorithm 1:
  - for i=1:n, for j=1:n, A(i,j) = B(i,j) + C(i,j)
- Algorithm 2:
  - for j=1:n, for i=1:n, A(i,j) = B(i,j) + C(i,j)
- What is “memory access pattern” for Algs 1 and 2?
- Which is faster?
- What if A, B, C stored by row (as in C)?

41

Using a Simpler Model of Memory to Optimize

- Assume just 2 levels in the hierarchy, fast and slow
- All data initially in slow memory
  - m = number of memory elements (words) moved between fast and slow memory
  - tm = time per slow memory operation
  - f = number of arithmetic operations
  - tf = time per arithmetic operation < tm
  - q = f/m average number of flops per slow element access
- Minimum possible Time = f*tf, when all data in fast memory
- Actual Time = f*tf + m*tm = f*tf*(1 + (tm/tf)*(1/q))
- Larger q means Time closer to minimum f*tf

42

Simple example using memory model

- To see results of changing q, consider simple computation

```
s = 0
for i = 1, n
  s = s + h(X[i])
```

- Assume tf=1 Mflop/s on fast memory
- Assume moving data is tm = 10
- Assume h takes q flops
- Assume array X is in slow memory

- So m = n and f = q*n
- Time = read X + compute = 10*n + q*n
- Mflop/s = f/t = q/(10 + q)
- As q increases, this approaches the "peak" speed of 1 Mflop/s

43

Simple Example (continued)

- Algorithm 1

```
s1 = 0;  s2 = 0
for j = 1 to n
    s1 = s1+h1(X(j))
    s2 = s2 + h2(X(j))
```

° Algorithm 2

```
s1 = 0; s2 = 0
for j = 1 to n
    s1 = s1 + h1(X(j))
for j = 1 to n
    s2 = s2 + h2(X(j))
```

° Which is faster?

44

Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
      a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
      d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {   a[i][j] = 1/b[i][j] * c[i][j];
      d[i][j] = a[i][j] + c[i][j];}
```

2 misses per access to a & c vs. one miss per access; improve spatial locality

45

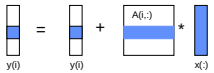
Optimizing Matrix Multiply for Caches

- Several techniques for making this faster on modern processors
  - heavily studied
- Some optimizations done automatically by compiler, but can do much better
- In general, you should use optimized libraries (often supplied by vendor) for this and other very common linear algebra operations
  - BLAS = Basic Linear Algebra Subroutines
- Other algorithms you may want are not going to be supplied by vendor, so need to know these techniques

46

Warm up: Matrix-vector multiplication y = y + A*x

```
for i = 1:n
    for j = 1:n
        y(i) = y(i) + A(i,j)*x(j)
```



47

Warm up: Matrix-vector multiplication y = y + A*x

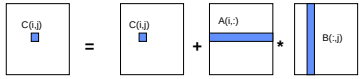
```
{read x(1:n) into fast memory}
{read y(1:n) into fast memory}
for i = 1:n
    {read row i of A into fast memory}
    for j = 1:n
        y(i) = y(i) + A(i,j)*x(j)
{write y(1:n) back to slow memory}
```

° m = number of slow memory refs = 3*n + n^2
° f = number of arithmetic operations = 2*n^2
° q = f/m ~= 2
° Matrix-vector multiplication limited by slow memory speed

48

Matrix Multiply C=C+A*B

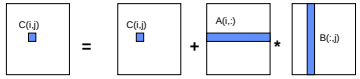
```
for i = 1 to n
    for j = 1 to n
        for k = 1 to n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
```



49

Matrix Multiply C=C+A*B(unblocked, or untiled)

```
for i = 1 to n
 {read row i of A into fast memory}
  for j = 1 to n
    {read C(i,j) into fast memory}
    {read column j of B into fast memory}
    for k = 1 to n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    {write C(i,j) back to slow memory}
```



50

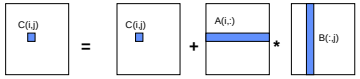
Matrix Multiply (unblocked, or untiled)

q=ops/slow mem ref

Number of slow memory references on unblocked matrix multiply

m = n^3 read each column of B n times
+ n^2 read each column of A once for each i
+ 2*n^2 read and write each element of C once
= n^3 + 3*n^2

So q = f/m = (2*n^3)/(n^3 + 3*n^2)
~= 2 for large n, no improvement over matrix-vector mult

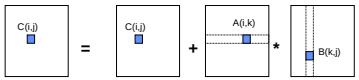


51

Matrix Multiply (blocked, or tiled)

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

```
for i = 1 to N
    for j = 1 to N
        {read block C(i,j) into fast memory}
        for k = 1 to N
            {read block A(i,k) into fast memory}
            {read block B(k,j) into fast memory}
            C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
        {write block C(i,j) back to slow memory}
```



52

Matrix Multiply (blocked or tiled)

q=ops/slow mem ref

Why is this algorithm correct?

Number of slow memory references on blocked matrix multiply
m = N*n^2 read each block of B N^3 times (N^3 * n/N * n/N)
+ N*n^2 read each block of A N^3 times
+ 2*n^2 read and write each block of C once
= (2*N + 2)*n^2
So q = f/m = 2*n^3 / ((2*N + 2)*n^2)
~= n/N = b for large n

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

Limit: All three blocks from A,B,C must fit in fast memory (cache), so we cannot make these blocks arbitrarily large: 3*b^2 <= M, so q ~= b <= sqrt(M/3)

Theorem (Hong, Kung, 1981): Any reorganization of this algorithm (that uses only associativity) is limited to q =O(sqrt(M))

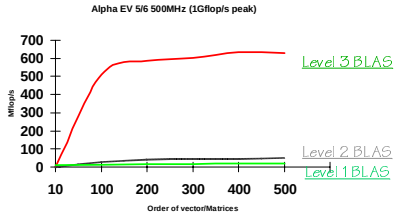
53

More on BLAS (Basic Linear Algebra Subroutines)

- Industry standard interface(evolving)
- Vendors, others supply optimized implementations
- History
  - BLAS1 (1970s):
    - » vector operations: dot product, saxpy (y=α*x+y), etc
    - » m=2*n, f=2*n, q ~1 or less
  - BLAS2 (mid 1980s)
    - » matrix-vector operations: matrix vector multiply, etc
    - » m=n^2, f=2*n^2, q~2, less overhead
    - » somewhat faster than BLAS1
  - BLAS3 (late 1980s)
    - » matrix-matrix operations: matrix matrix multiply, etc
    - » m >= 4n^2, f=O(n^3), so q can possibly be as large as n, so BLAS3 is potentially much faster than BLAS2
- Good algorithms used BLAS3 when possible (LAPACK)
- www.netlib.org/blas, www.netlib.org/lapack

54

BLAS for Performance



- **Development of blocked algorithms important for performance**

BLAS 3 (n-by-n matrix matrix multiply) vs
BLAS 2 (n-by-n matrix vector multiply) vs
BLAS 1 (saxpy of n vectors)

Optimizing in practice

- Tiling for registers
  - loop unrolling, use of named "register" variables
- Tiling for multiple levels of cache
- Exploiting fine-grained parallelism within the processor
  - super scalar
  - pipelining
- Complicated compiler interactions
- Hard to do by hand (but you'll try)
- Automatic optimization an active research area
  - PHIPAC: www.icsi.berkeley.edu/~bilmes/phipac
  - www.cs.berkeley.edu/~iyer/asci_slides.ps
  - ATLAS: www.netlib.org/atlas/index.html

Strassen's Matrix Multiply

- The traditional algorithm (with or without tiling) has O(n^3) flops
- Strassen discovered an algorithm with asymptotically lower flops
  - O(n^2.81)
- Consider a 2x2 matrix multiply, normally 8 multiplies

Let M = [m11 m12] = [a11 a12] * [b11 b12]
[m21 m22] [a21 a22] [b21 b22]

Let p1 = (a12 - 122) * (b21 + b22)
p2 = (a11 + a22) * (b11 + b22)
p3 = (a11 - a21) * (b11 + b12)
p4 = (a11 + a12) * b22
p5 = a11 * (b12 - b22)
p6 = a22 * (b21 - b11)
p7 = (a21 + a22) * b11

Then m11 = p1 + p2 - p4 + p6
m12 = p4 + p5
m21 = p6 + p7
m22 = p2 - p3 + p5 - p7

Extends to nxn by divide&conquer

Strassen (continued)

T(n) = Cost of multiplying nxn matrices
= 7*T(n/2) + 18*(n/2)^2
= O(n^log_2 7)
= O(n^2.81)

° Available in several libraries
° Up to several time faster if n large enough (100s)
° Needs more memory than standard algorithm
° Can be less accurate because of roundoff error
° Current world's record is O(n^2.376..)

Summary

- Performance programming on uniprocessors requires
  - understanding of memory system
    - » levels, costs, sizes
  - understanding of fine-grained parallelism in processor to produce good instruction mix
- Blocking (tiling) is a basic approach that can be applied to many matrix algorithms
- Applies to uniprocessors and parallel processors
  - The technique works for any architecture, but choosing the blocksize b and other details depends on the architecture
- Similar techniques are possible on other data structures
- You will get to try this in Assignment 2 (see the class homepage)

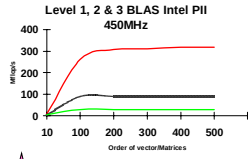
Summary: Memory Hierachy

- Virtual memory was controversial at the time: can SW automatically manage 64KB across many programs?
  - 1000X DRAM growth removed the controversy
- Today VM allows many processes to share single memory without having to swap all processes to disk; today VM protection is more important than memory hierarchy
- Today CPU time is a function of (ops, cache misses) vs. just f(ops): What does this mean to Compilers, Data structures, Algorithms?

Performance = Effective Use of Memory Hierarchy

- **Can only do arithmetic on data at the top of the hierarchy**
- **Higher level BLAS lets us do this**

BLAS	Memory Refs	Flops	Flops/ Memory Refs
Level 1 y=y+ax	$3n$	$2n$	2/3
Level 2 y=y+Ax	n^2	$2n^2$	2
Level 3 C=C+AB	$4n^2$	$2n^3$	n/2



- **Development of blocked algorithms important for performance**

61

Homework Assignment

- Implement, in Fortran or C, the six different ways to perform matrix multiplication by interchanging the loops. (Use 64-bit arithmetic.) Make each implementation a subroutine, like:

- **subroutine ijk (a, m, n, lda, b, k, ldb, c, ldc)**
- **subroutine ikj (a, m, n, lda, b, k, ldb, c, ldc)**
- **…**

62

Thanks

- **These slides came in part from courses taught by the following people:**
  - **Kathy Yelick, UC, Berkeley**
  - **Dave Patterson, UC, Berkeley**
  - **Randy Katz, UC, Berkeley**
  - **Craig Douglas, U of Kentucky**
- **Computer Architecture A Quantitative Approach, Chapter 8, Hennessy and Patterson, Morgan Kaufman Pub.**

63