February 13, 2025

Announcements

- HW1
- scheduling contest

Goals

- Bandwidth model of MM
- I/O cost
  - ↳ code structure?
- memory: missing aspects
- SIMD
- towards multi-core

Review

- Mem hierarchy
  - Caches
  - Streaming workloads
    - ↳ "nth"
    - ↳ fine-grain possible (e.g. PTX)
- MM latency accounting:
  - N, N_s, N_s
  - Miss rate $\sim \frac{1}{N_s}$

Case study: Matrix-Matrix Mult. ('MMM') via Bandwidth

Cost model for MMM in a two-level hierarchy based on bandwidth?

FMA ("fused multiply add") { 2 for the cost of 1 in terms of throughput only one rounding step

512 bit / 32 bit SP = 16x

~ ×2 for 2 VALUs = 32/cycle

Cycle count (multiply): N^3 FMAs / (32 FMAs/cycle) = $\frac{N^3}{32}$ cycles.

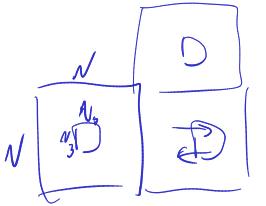
cache traffic / cycle: 4 operands · N^3 / cycle count = $\frac{4N^3}{N^3/32}$ = 128 $\frac{f}{s}$

$128 \frac{f}{c} \; 4 \frac{B}{f} \cdot 3 \cdot 10^9 \frac{c}{s} = 1536 \cdot 10^9$

[Yotov et al. '07]

$\sum a_{ij} \cdot b_{nj}$

Total cache ↔ DRAM traffic:



$\underbrace{(N/N_B)^3}_{\#\text{ tile calculations}} \cdot 4 \cdot N_B^2 = 4N^3/N_B$

100 GB/s

$\frac{2560\ \text{GB/s}}{N_B}$ cache bw $\geq$ DRAM bw

$N_B \times 25$ ⇝ cache size ~ 10kB

Case study: Matrix-Matrix Mult. ('MMM'): Discussion

Discussion: What are the main simplifications in each model?

[Yotov et al. '07]

General Q: How can we analyze cache cost of algorithms in general?

Hong/Kung: Red/Blue Pebble Game

Simple means of I/O cost analysis: "Red/blue pebble game"

- A way to quantify I/O cost on a DAG (why a DAG?)
- "Red Hot" pebbles: data that can be computed on
- "Blue Cool" pebbles: data that is stored, but not available for computation without I/O

Note: Can allow "Red/Purple/Blue/Black": more levels

Q: What are the cost metrics in this model?

[Hong/Kung '81]

$$\#\text{steps} \cdot \sqrt{\#\text{red pebbles}} = \Omega(n^3)$$

$$f(n) = \Omega(g(n))$$

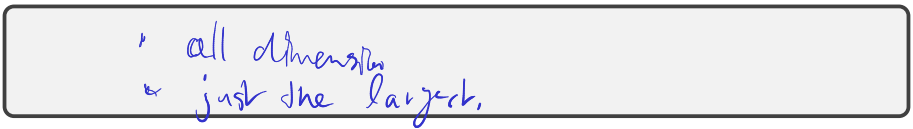
$$\Leftrightarrow g(n) = O(f(n)) \quad \text{(Knuth)}$$

Cache-Oblivious Algorithms

Annoying chore: Have to pick multiple machine-adapted block sizes in cache-adapted algorithms, one for each level in the memory hierarchy, starting with registers.

Idea:

- Step 1: Express algorithm recursively in divide & conquer manner
- Step 2: Pick a strategy to decrease block size
  Give examples of block size strategies, e.g. for MMM:



Result:

- Asymptotically optimal on Hong/Kung metric

Cache-Oblivious Algorithms: Issues

What are potential issues on actual hardware?

- Iterative more "design DOFs"
- function call overhead
- too many levels of blocking

call (loc)

ret

$n! = n \cdot (n-1)!$

[Yotov et al. '07]

Recall: Big-O Notation

Classical Analysis of Algorithms (e.g.):

$$\text{Cost}(n) = O(n^3).$$

Precise meaning? Anything missing from that statement?

$(n \to \infty)$

There exists a C and an N_0 so that for $n \geq N_0$

$$\text{Cost} \leq C \cdot n^3$$

Comment: "Asymptotically Optimal"

Comments on asymptotic statements about cost in relation to high performance?

- No statement about finite n
- No statement about the constant

Net effect: Having an understanding of asymptotic cost is *necessary*, but *not sufficient* for high performance.

HPC is in the business of minimizing C in:

$$\text{Cost}(n) \leq C \cdot n^3 \quad \text{(for all } n\text{)}$$

Alignment

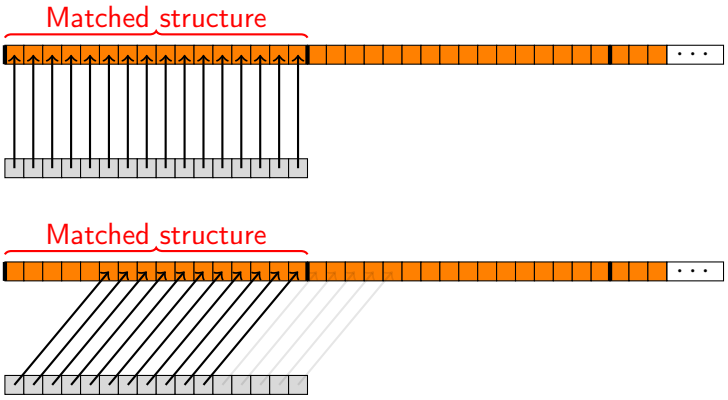
Alignment describes the process of matching the base address of:

- Single word: `double`, `float`
- SIMD vector
- Larger structure

To machine granularities:

Q: What is the performance impact of misalignment?

Performance Impact of Misalignment



SIMD: Basic Idea

What's the basic idea behind SIMD?

What architectural need does it satisfy?

Typically characterized by width of data path:

- SSE: 128 bit (4 floats, 2 doubles)
- AVX-2: 256 bit (8 floats, 4 doubles)
- AVX-512: 512 bit (16 floats, 8 doubles)

SIMD: Architectural Issues

Realization of inter-lane comm. in SIMD? Find instructions.

Name tricky/slow aspects in terms of expressing SIMD:

SIMD: Intel Instructions

x86 SIMD suffixes: What does the "`ps`" suffix mean? "`sd`"?

SIMD: Transposes

Why are transposes important? Where do they occur?

Example implementation aspects:

- HPTT: [Springer et al. '17]
- github: springer13/hptt 8x8 transpose microkernel
- Q: Why 8x8?

Outline

Multiple Cores vs Bandwidth

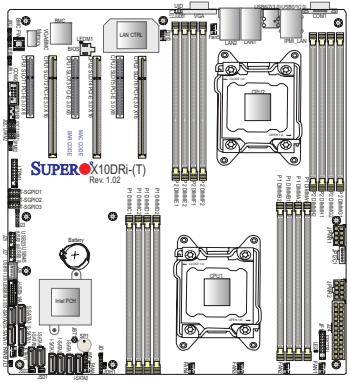
Assume (roughly right for Intel):

- memory latency of 100 ns
- peak DRAM bandwidth of 50 GB/s (per socket)

How many cache lines should be/are in flight at one time?

[McCalpin '18]

Topology and NUMA



[SuperMicro Inc. ‘15]

Demo: Show `lstopo` on porter, from hwloc.

Placement and Pinning

Who decides on what core my code runs? How?

Who decides on what NUMA node memory is allocated?

Demo: intro/NUMA and Bandwidths

What is the main expense in NUMA?

Cache Coherence

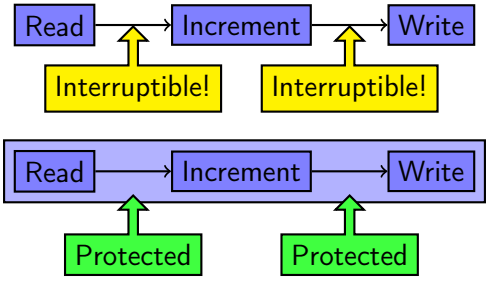
What is *cache coherence*?

How is cache coherence implemented?

What are the performance impacts?

- Demo: intro/Threads vs Cache
- Demo: intro/Lock Contention

'Conventional' vs Atomic Memory Update



Outline

Outline

Atomic Operations: Compare-and-Swap

```
#include <stdatomic.h>
_Bool atomic_compare_exchange_strong(
      volatile A* obj, C* expected, C desired );
```

What does `volatile` mean?

What does this do?

How might you use this to implement atomic FP multiplication?

Memory Ordering

Why is Memory Ordering a Problem?

What are the different memory orders and what do they mean?

Example: A Semaphore With Atomics

```
#include <stdatomic.h> // mo_->memory_order, a_->atomic
typedef struct { atomic_int v;} naive_sem_t;
void sem_down(naive_sem_t *s)
{
  while (1) {
    while (a_load_explicit(&(s->v), mo_______) < 1)
        spinloop_body();
    int tmp=a_fetch_add_explicit(&(s->v), -1, mo_________rel);
    if (tmp >= 1)
        break; // we got the lock
    else // undo our attempt
      a_fetch_add_explicit(&(s->v), 1, mo_______);
  }
}
void sem_up(naive_s_t *s) {
  a_fetch_add_explicit(&(s->v), 1, mo_______);
}
```

[Cordes '16] — Hardware implementation: how?

C: What is 'order'?

C11 Committee Draft, December '10, Sec. 5.1.2.3, "Program execution":

- (3) *Sequenced before* is an asymmetric, transitive, pair-wise relation between evaluations executed by a single thread, which induces a partial order among those evaluations. Given any two evaluations A and B, if A is sequenced before B, then the execution of A shall precede the execution of B. (Conversely, if A is sequenced before B, then B is sequenced after A.) If A is not sequenced before or after B, then A and B are unsequenced. Evaluations A and B are *indeterminately sequenced* when A is sequenced either before or after B, but it is unspecified which. The presence of a sequence point between the evaluation of expressions A and B implies that every value computation and side effect associated with A is sequenced before every value computation and side effect associated with B. (A summary of the *sequence points* is given in annex C.)

Q: Where is this definition used (in the standard document)?

C: What is 'order'? (Encore)

C11 Draft, 5.1.2.4 "Multi-threaded executions and data races":

- All modifications to a particular atomic object M occur in some particular total order, called the *modification order* of M.
- An evaluation A *carries a dependency* to an evaluation B if ...
- An evaluation A is *dependency-ordered* before an evaluation B if...
- An evaluation A *inter-thread happens before* an evaluation B if...
- An evaluation A *happens before* an evaluation B if...

Why is this so subtle?

C: How Much Lying is OK?

C11 Committee Draft, December '10, Sec. 5.1.2.3, "Program execution":

- (1) The semantic descriptions in this International Standard describe the behavior of an abstract machine in which issues of optimization are irrelevant.
- (2) Accessing a volatile object, modifying an object, modifying a file, or calling a function that does any of those operations are all *side effects*, which are changes in the state of the execution environment. [...]

C: How Much Lying is OK?

- (4) In the abstract machine, all expressions are evaluated as specified by the semantics. An actual implementation need not evaluate part of an expression if it can deduce that its value is not used and that no needed side effects are produced (including any caused by calling a function or accessing a volatile object).
- (6) The least requirements on a conforming implementation are:
  - Accesses to volatile objects are evaluated strictly according to the rules of the abstract machine.
  - At program termination, all data written into files shall be identical to the result that execution of the program according to the abstract semantics would have produced.
  - The input and output dynamics of interactive devices shall take place as specified in 7.21.3. The intent of these requirements is that unbuffered or line-buffered output appear as soon as possible, to ensure that prompting messages actually appear prior to a program waiting for input.

  This is the observable behavior of the program.

Arrays

Why are arrays the dominant data structure in high-performance code?

Any comments on C's arrays?

Arrays vs Abstraction

Arrays-of-Structures or Structures-of-Arrays? What's the difference? Give an example.

Language aspects of the distinction? Salient example?

C and Multi-Dimensional Arrays: A Saving Grace

```
// YES:
void f(int m, int n, double (*)[m][n]);

// NO:
struct ary {
  int m;
  int n;
  double (*array)[m][n];
};

// YES:
struct ary {
  int m;
  int n;
  double a[];
};
```