CS 6354: Memory Hierarchy III

5 September 2016

Naïve (1)

```
for (int i = 0; i < I; ++i) {
  for (int j = 0; j < J; ++j) {
    for (int k = 0; k < K; ++k) {
      C[i * K + k] +=
        A[i * J + j] * B[j * K + k];
    }
  }
}
```

Naïve (1)

```
for (int i = 0; i < I; ++i) {
  for (int j = 0; j < J; ++j) {
    for (int k = 0; k < K; ++k) {
      C[i * K + k] +=
        A[i * J + j] * B[j * K + k];
    }
  }
}
```

Naïve (2)

```
for (int i = 0; i < I; ++i) {
  for (int k = 0; k < K; ++k) {
    for (int j = 0; j < J; ++j) {
      C[i * K + k] +=
        A[i * J + j] * B[j * K + k];
    }
  }
}
```

Naïve (2)

```
for (int i = 0; i < I; ++i) {
  for (int k = 0; k < K; ++k) {
    for (int j = 0; j < J; ++j) {
      C[i * K + k] +=
        A[i * J + j] * B[j * K + k];
    }
  }
}
```

Goto Fig. 4

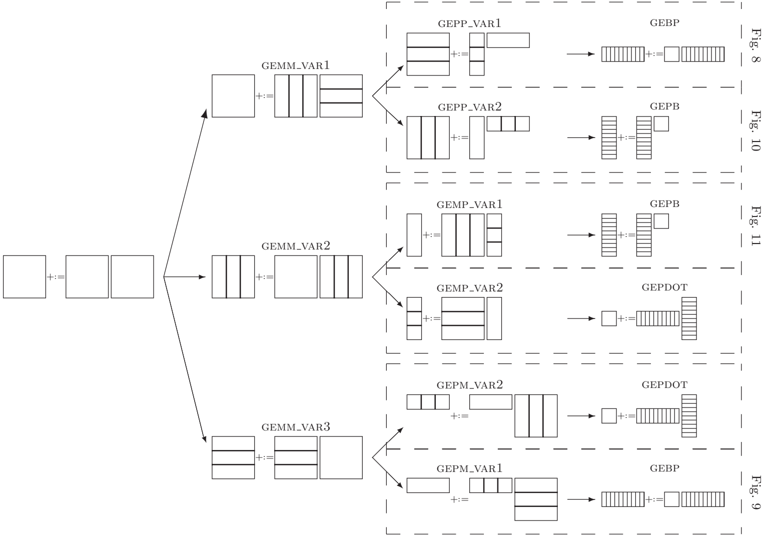


Fig. 4. Layered approach to implementing GEMM.

The Inner Loop

```
for p = 1 : K

  for i = 1 : M

    for j = 1 : N
      C_ij += A_ip B_pj
    endfor

  endfor

endfor
```

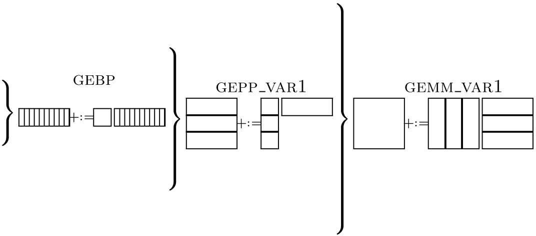
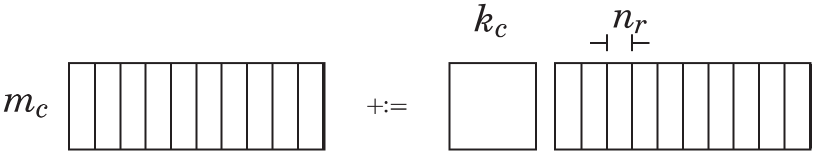


Fig. 5. The algorithm that corresponds to the path through Figure 4 that always takes the top branch expressed as a triple-nested loop.



GFLOP/s

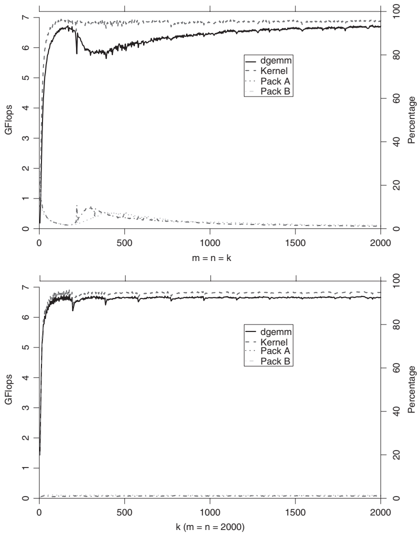


Fig. 14. Pentium4 Prescott (3.6 GHz).

Theoretical maximum performance

This CPU: 2 double adds or multiplies per cycle

3.6 GHz: 7.2B adds or multiplies per second

= 7.2 Gflop/s (Giga Floating Point Operation Per Second)

Theme: Overlap

Modern CPUs do other things during memory operations

ideal: no added latency

Cache/Register Blocking

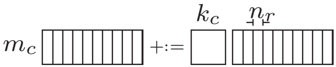
minimize data movements

... by reordering computation

best orders — all computations within ‘block’

Load into Cache?

Algorithm: $C := \textsc{gebp}(A, B, C)$



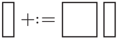
Load A into cache	($m_c k_c$ memops)
for $j = 0, \ldots, N-1$	
Load B_j into cache	($k_c n_r$ memops)
Load C_j into cache	($m_c n_r$ memops)
	($C_j := AB_j + C_j$)
Store C_j into memory	($m_c n_r$ memops)
endfor	

Fig. 7. Basic implementation of GEBP.

Why packing?

250 x ??? matrix at memory address 300, working on first part:

300	301	302	303	304	305	306	307	308	309	310	311	…	549
550	551	552	553	554	555	556	557	558	559	560	561	…	799
800	801	802	803	804	805	806	807	808	809	810	811	…	1049
1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	…	1299
1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	…	1549

Why packing?

250 x ??? matrix at memory address 300, working on first part:

300	301	302	303	304	305	306	307	308	309	310	311	…	549
550	551	552	553	554	555	556	557	558	559	560	561	…	799
800	801	802	803	804	805	806	807	808	809	810	811	…	1049
1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	…	1299
1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	…	1549

unused parts of cache blocks

- irrelevant 310 in same block as 309

Why packing?

250 x ??? matrix at memory address 300, working on first part:

300	301	302	303	304	305	306	307	308	309	310	311	…	549
550	551	552	553	554	555	556	557	558	559	560	561	…	799
800	801	802	803	804	805	806	807	808	809	810	811	…	1049
1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	…	1299
1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	…	1549

unused parts of cache blocks

- irrelevant 310 in same block as 309

conflict misses if close-to-power-of-two

- nearby matrix entries map to same set

Why packing?

250 x ??? matrix at memory address 300, working on first part:

300	301	302	303	304	305	306	307	308	309	310	311	…	549
550	551	552	553	554	555	556	557	558	559	560	561	…	799
800	801	802	803	804	805	806	807	808	809	810	811	…	1049
1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	…	1299
1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	…	1549

unused parts of cache blocks

- irrelevant 310 in same block as 309

conflict misses if close-to-power-of-two

- nearby matrix entries map to same set

extra TLB misses

- less of relevant matrix in each page

The Balanced System

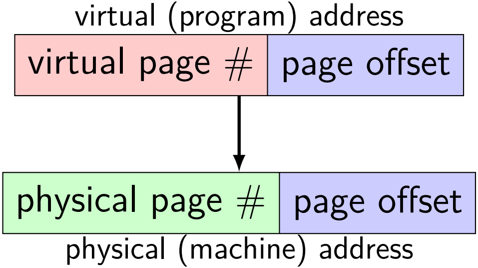
$$n_r \geq \frac{R_{\text{comp}}}{2R_{\text{load}}}$$

$C = AB$

overlap loads (at rate R_{load}) from L2 with computation

enough of C, B (n_r) in L1/registers to keep FPU busy

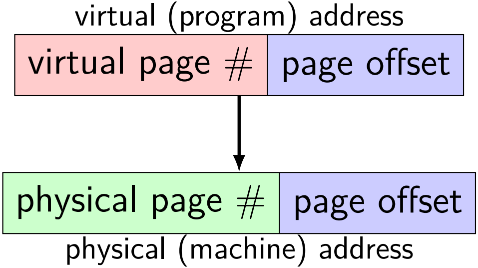
TLB capacities



TLB (cache of page table)

virtual	**physical**
0x00444	0x007
0x00446	0x01c
0x00448	0x01f
0x0044a	0x024

TLB capacities

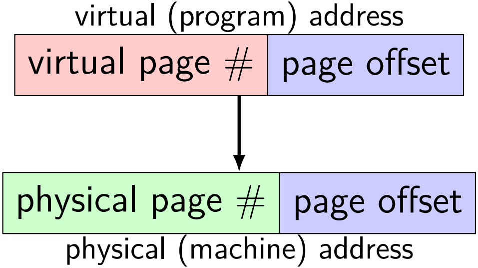


TLB (cache of page table)

virtual	physical
0x00444	0x007
0x00446	0x01c
0x00448	0x01f
0x0044a	0x024

reach: page size × # entries = 16K with 4K pages

TLB capacities



TLB (cache of page table)

virtual	**physical**
`0x00444`	`0x007`
`0x00446`	`0x01c`
`0x00448`	`0x01f`
`0x0044a`	`0x024`

reach: page size $\times$ # entries $=$ 16K with 4K pages

worst case: each entry only useful for 1 byte of data:

e.g. `0x00444ccc` `0x00446bbb` `0x00448aaa` `0x0044a999` `0x0044c777` etc.

Hierarchical page tables

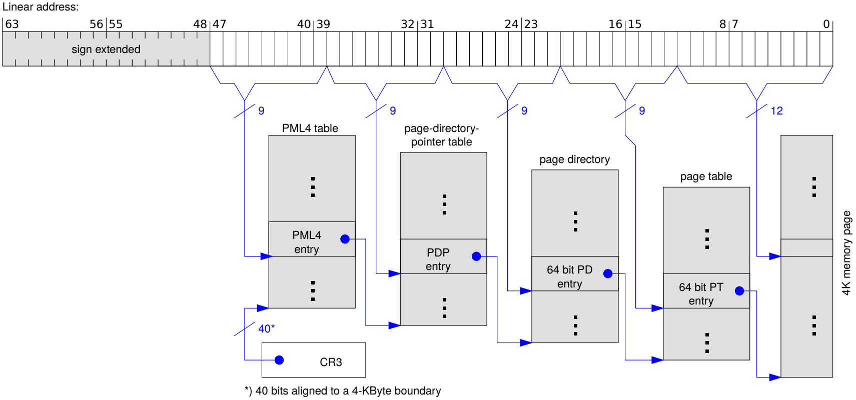


Diagram: Wikimedia / RokerHRO

Large pages (1)

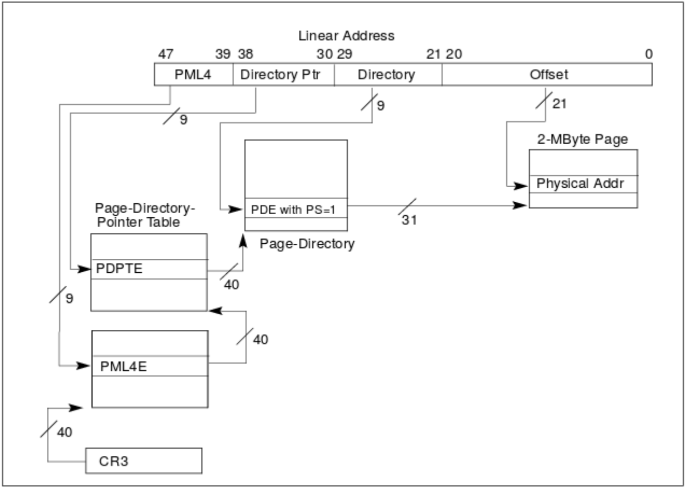


Diagram: Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3A

Large pages (2)

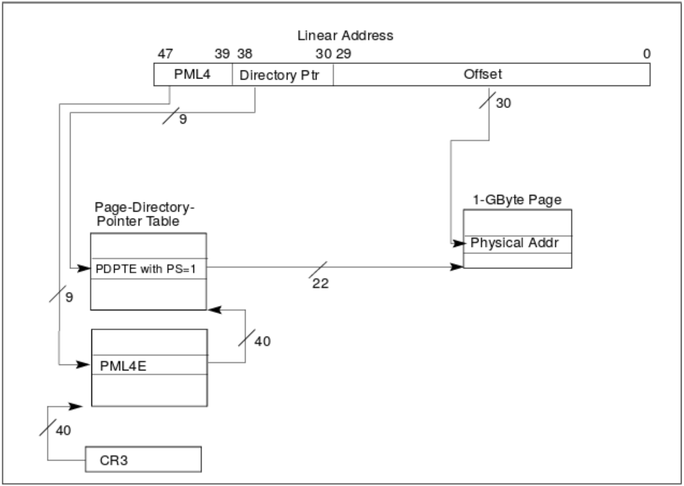


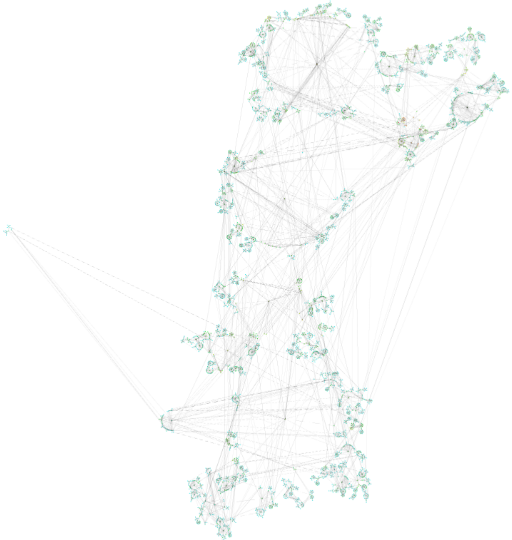
Diagram: Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3A

Data TLB reach on my laptop

4KB pages:	64 pages =	256 KB
2M pages:	32 pages =	64 MB
1GB pages:	4 pages =	4 GB

256 KB — smaller than L3 cache

Intuition: why no locality



Amazon recommendation network from Lehmann and Kottler, “Visualizing Large and Clustered Networks”

Proof of locality?

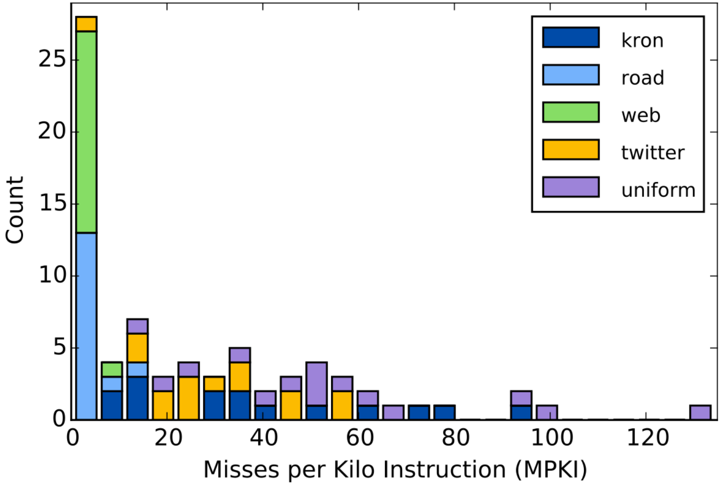


Fig. 7. Single-thread MPKI (in terms of LLC misses) of full workload.

Preview: Out-of-order

What happens on a cache miss?

Preview: Out-of-order

What happens on a cache miss?

modern fast CPUs: keep executing instructions

...unless value actually needed

Preview: Reorder buffer

holds pending instructions

used to make computation appear in-order

(more later in the course)

key feature here: need to have enough room for every instruction run out-of-order

Non-Uniform Memory Access

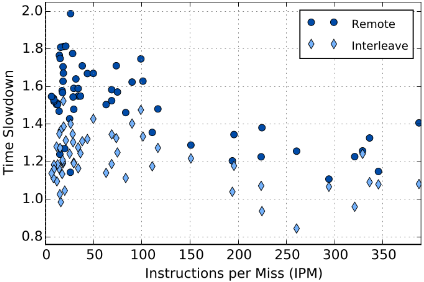
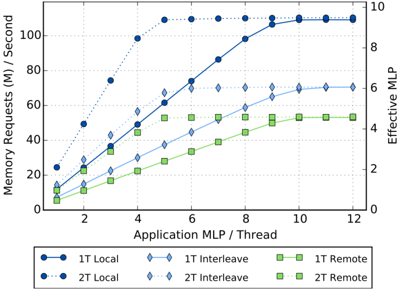


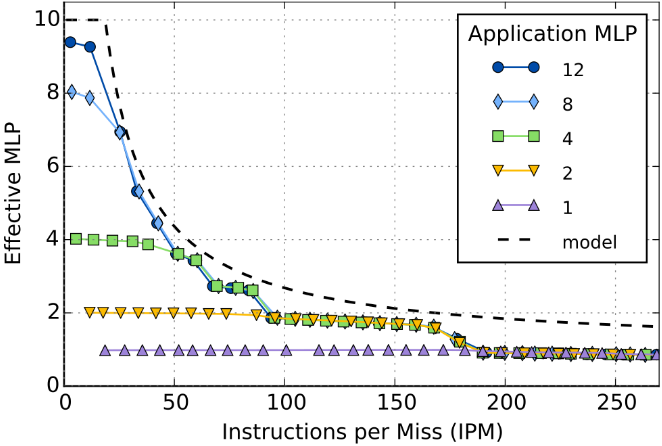
Fig. 12. Single-socket (8 cores) slowdown relative to local memory of full workload executing out of remote memory or interleaved memory.



Some memory closer to one core than another

Exists within a socket (single chip)

Memory Request Limits



Page table overhead

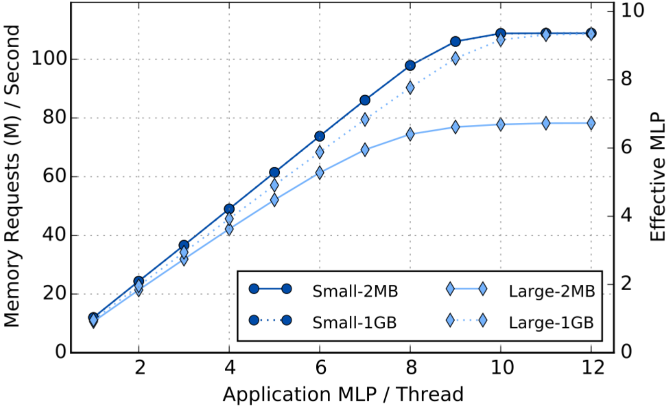


Fig. 3. Impact of 2 MB and 1 GB page sizes on memory bandwidth achieved by single-thread parallel pointer chase for array sizes of *small* (1 GB) and *large* (16 GB).

Pointer chasing

```
void **pointer = /* initialize array */;
for (int i = 0; i < MAX_ITER; ++i) {
    pointer = *pointer;
}
```

Preview: SMT

What happens on cache miss?

Run a different thread!

Needs: extra set of registers

Same machinary as out-of-order

(more later in the course)

Beamer's theory about SMT

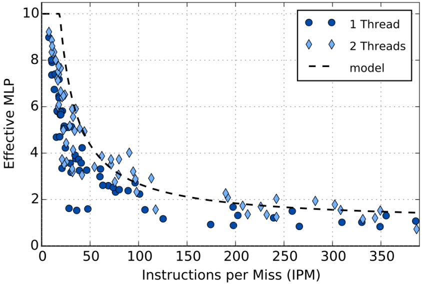


Fig. 15. Achieved memory bandwidth of full workload relative to instructions per miss (IPM) with one or two threads on one core.

"One thread could generate most of the cache misses sustaining a high effective MLP while the other thread (unencumbered by cache misses) could execute instructions quickly to increase IPM."

"In practice, the variation between threads is modest…"

Conditions

Any LLC miss will cause even a large out-of-order processor to stall for a significant number of cycles. Ideally, while waiting for the first cache miss to resolve, at least some useful work could be done, including initiating loads early that will cause future cache misses. Unfortunately, a load must satisfy the following four requirements before it can be issued:

1) *Processor fetches load instruction* - Control flow reaches the load instruction (possibly speculatively).
2) *Space in instruction window* - The Reorder Buffer (ROB) is not full and has room for the load.
3) *Register operands are available* - The load address can be generated.
4) *Memory bandwidth is available* - At the core level there is a miss-status holding register (MSHR) available and there is not excessive contention within the on-chip interconnect or at the memory controller.

Where to do graph processing?

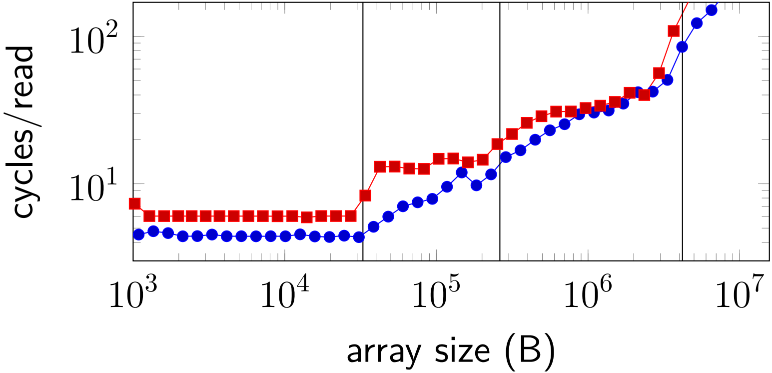
Extreme: Cray XMT

- no data cache
- 100s of outstanding memory acccesses (“memory-level parallelism”)

Homework 1

Example: measure sizes of each data/unified cache

Benchmark: speed of accessing array of varying size in random order



Note on Paper Reviews (1)

Make it clear where you answer each part

- You can copy-and-paste the questions

Only need one significant insight

- Better to explain one well (including evidence) than three poorly

Your insight should be a result

- What experiments showed, not what experiments were done

Note on Paper Reviews (2)

Evidence: not just that there were experiments

- What kind of experiments?
- How big is the effect?

Weakness/improvement: don't be afraid

- Often the discussion identifies these for you

Next time

“Performance from architecture: comparing a RISC and CISC with similar hardware organization”

- CISC (VAX) v RISC (MIPS)
- both pipelined
- microinstructions to implement complex instructions

”The RISC V Instruction Set Manual: Volume I: User-Level ISA”, Chapter 1 (including commentary) only

- motivation (chapter 1 only) for a recent ISA design