RED HAT :: BOSTON :: 2008

SUMMIT

JUNE 18–20, 2008

Understanding Application Memory Performance

Ulrich Drepper
Consulting Engineer, Red Hat

Why Singling Out Memory?

- Speed of Computer Main Memory does not keep up

	Clock	Memory Access	Effective Clock
early 80s	1MHz	1 cycle	1 MHz
today	4GHz	250 cycles	16 Mhz

- Memory cannot get much faster, latency-wise

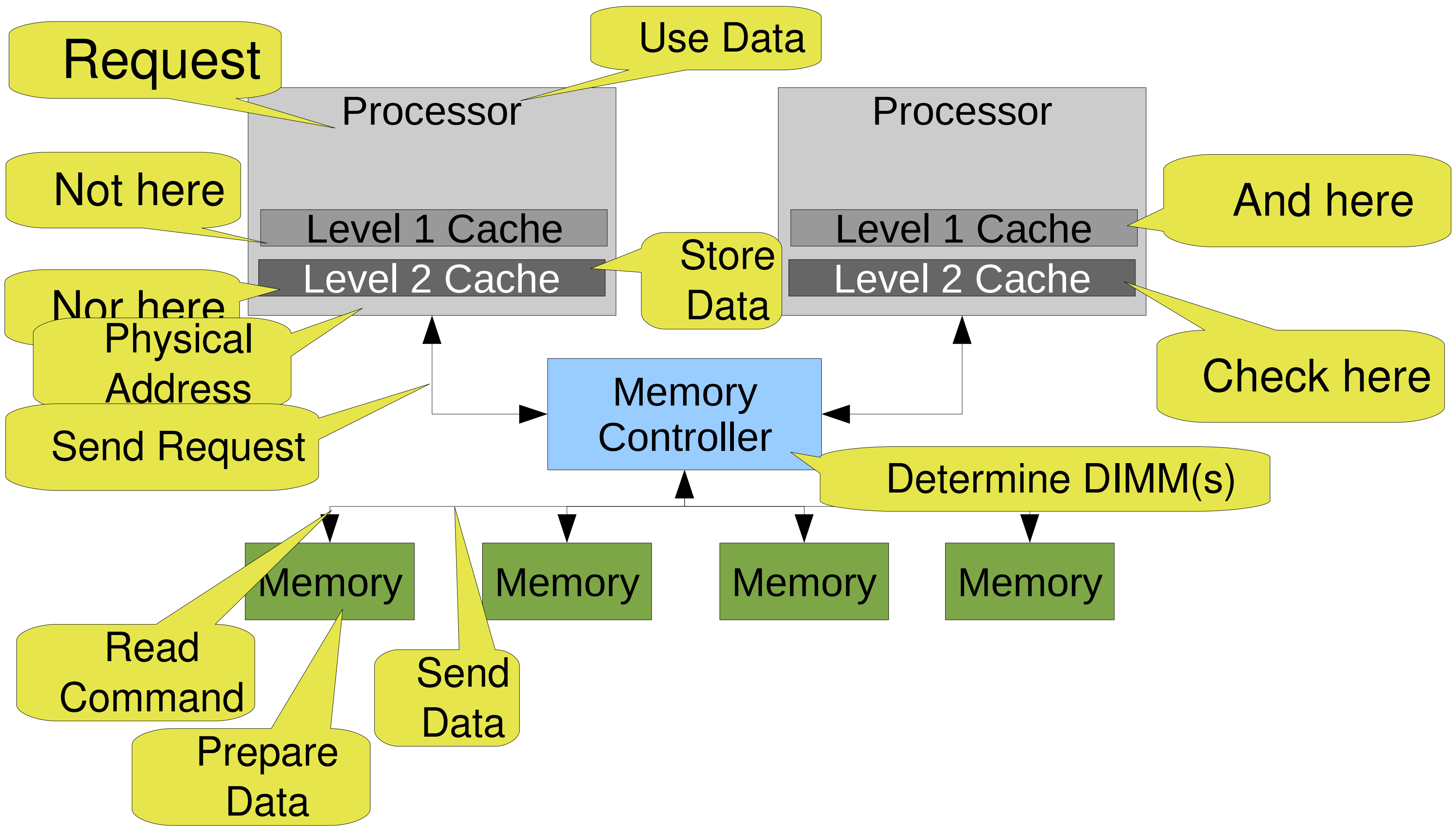
$$\text{Energy} = \text{Capacity} \cdot \text{Voltage}^2 \cdot \text{Frequency}$$

- Increased competition for memory connection due to many-core processors

Why Is Memory Performance Optimization Hard?

- Memory technology not well understood
- There are so many places where memory is accessed
- Effects not local
  - Entire program should be understood for best results
  - Other processes can have effects, too
- Hardware Complications
  - Multi-core
  - NUMA

Overview


Request
Use Data
Processor
Processor
Not here
Level 1 Cache
Level 1 Cache
And here
Store Data
Nor here
Level 2 Cache
Level 2 Cache
Physical Address
Check here
Memory Controller
Send Request
Determine DIMM(s)
Memory
Memory
Memory
Memory
Read Command
Send Data
Prepare Data

Important Factors

- Cache Line Utilization
- Memory Page Utilization
- TLB Branch Utilization
- Avoid just-in-time reading:
  - Help hardware prefetching
  - Use explicit software prefetching
- Parallelism
  - Concurrent cache-line use
  - Frequent cache-line transfer
- Non-local access

An Example: Matrix Multiplication

```
for (size_t i = 0; i < X; ++i)
  for (size_t j = 0; j < Z; ++j)
    for (size_t k = 0; k < Y; ++k)
      res[i][j] += mul1[i][k] * mul2[k][j];
```

Both matrixes have size 2048x2048

- 8,589,934,592 multiplications and additions
- 3GHz Intel Core2
- Runtime: 678 sec!
- 12,669,520 FLOPS

Measure!

Oprofile: statistical profiling

- Use hardware performance counters (10 sec each)

Event	Count	Event	Count
CPU_CLK_UNHALTED	5,302,632,000	L1D_REPL	183,174,500
INST_RETIRED	435,096,000	L2_LINES_IN.ANY	240,435,000
RESOURCE_STALLS	1,886,790,000	L2_LINES_IN.DEMAND	126,758,500
IFU_MEM_STALL	262,414,500	PAGE_WALKS	154,154,000
ITLB_MISS_RETIRED	28,500	DTLB_MISSES.ANY	118,965,000
L1I_MISSES	253,500	DTLB_MISSES.MISS_LD	131,460,500
L2_IFETCH	20,000	DTLB_MISSES.MISS_ST	24,500
L1D_CACHE_LD	139,074,500	L1D_CACHE_LD	177,222,500
STORE_BLOCK	141,500	L1D_CACHE_ST	122,000

What does each number mean?

Relativity

- Absolute numbers hard to interpret
- Create ratios (appendix B, Intel Optimization Manual)
- Ratios are independent of length of sampling
- No universal levels for ratios:
  - Memory-intensive code has more cache misses
  - Arithmetic-intensive code with have less, but more dependencies

Important Ratios

- Clocks per Instruction Retired

  `CPU_CLK_HALTED/INST_RETIRED`

  In multi-scalar processors, optimum > 1
- Instruction Fetch Stall

  CYCLES_L1I_MEM_STALLED/CPU_CLK_HALTED

  Any stall bad. Code should be predictable
- Virtual Table Use

  BR_IND_CALL_EXEC/INST_RETIRED

  Possible reason for instruction fetch stalls: indirect calls

Important Ratios

- Load Rate:

  L1D_CACHE_LD.MESI/CPU_CLK_UNHALTED

  Large number of loads means load/store buffers full all the time

- Store Order Block

  STORE_BLOCK.ORDER/CPU_CLK_UNHALTED

  Ratio of cycles in which instructions are held up because of write ordering due to cache misses

- L1 Data Cache Miss Rate

  L1D_REPL/INST_RETIRED

  How many instructions cause L1 cache misses

Important Ratios

- L2 Cache Miss Rate

  L2_LINES_IN/INST_RETIRED

  Instructions which cause L2 misses
- TLB Miss Penalty

  PAGE_WALKS/CPU_CLK_UNHALTED

  Cycles spent waiting for page table walks
- DTLB Miss Rate

  DTLB_MISSES/INST_RETIRED

  Instructions which cause DTLB misses

Ratios for the Example

- Some of the memory-related ratios:

CPU_CLK_UNHALTED / INST_RETIRED	12.19
RESOURCE_STALLS.RS_FULL / CPU_CLK_UNHALTED	77.36%
IFU_MEM_STALL / CPU_CLK_UNHALTED	8.84%
L1D_CACHE_LD / CPU_CLK_UNHALTED	0.03
L1D_REPL / INST_RETIRED	15.30%
L2_LINES_IN.ANY / INST_RETIRED	19.20%
L2_LINES_IN.DEMAND / INST_RETIRED	9.60%
PAGE_WALKS / CPU_CLK_UNHALTED	4.12%
DTLB_MISSES.MISS_LD / INST_RETIRED	9.90%
DTLB_MISSES.MISS_ST / INST_RETIRED	0.00%
L1D_CACHE_LD / INST_RETIRED	14.31%
L1D_CACHE_ST / INST_RETIRED	0.01%

Slightly Revised: Matrix Multiplication

```
for (size_t k = 0; k < Y; ++k)
  for (size_t j = 0; j < Z; ++j)
    for (size_t i = 0; i < X; ++i)
      res[i][j] += mul1[i][k] * mul2[k][j];
```

Swapped

- Now: 38 sec, 94% faster!

Visible Improvement

		Improvement
CPU_CLK_UNHALTED / INST_RETIRED	1.4	88.56%
RESOURCE_STALLS.RS_FULL / CPU_CLK_UNHALTED	8.87%	88.54%
IFU_MEM_STALL / CPU_CLK_UNHALTED	8.97%	-1.47%
L1D_CACHE_LD / CPU_CLK_UNHALTED	0.26	-814.29%
L1D_REPL / INST_RETIRED	15.39%	-0.59%
L2_LINES_IN.ANY / INST_RETIRED	1.32%	93.15%
L2_LINES_IN.DEMAND / INST_RETIRED	0.08%	99.17%
PAGE_WALKS / CPU_CLK_UNHALTED	0.53%	87.19%
DTLB_MISSES.MISS_LD / INST_RETIRED	0.03%	99.75%
DTLB_MISSES.MISS_ST / INST_RETIRED	0.02%	-2200.00%
L1D_CACHE_LD / INST_RETIRED	6.46%	54.88%
L1D_CACHE_ST / INST_RETIRED	0.71%	-7010.00%

Use Huge Pages

- mount hugetlbfs at `/mnt/huge`
- Use mmap with file descriptor for file under `/mnt/huge`

		Improvement
CPU_CLK_UNHALTED / INST_RETIRED	1.32	5.73%
DTLB_MISSES.MISS_LD / INST_RETIRED	0.02%	8.00%
DTLB_MISSES.MISS_ST / INST_RETIRED	0.02%	4.35%

Tiling

Fill in entire cache lines before they are evicted:

```
#define SM (64 / sizeof (double))
for (i = 0; i < X; i += SM)
  for (j = 0; j < Z; j += SM)
    for (k = 0; k < Y; k += SM)
      for (i2 = 0, rres = &RES(i, j), rmul1 = &MUL1(i, k); i2 < SM;
           ++i2, rres += Y, rmul1 += X)
        for (k2 = 0, rmul2 = &MUL2(k, j); k2 < SM; ++k2, rmul2 += Z)
          for (j2 = 0; j2 < SM; ++j2)
            rres[j2] += rmul1[k2] * rmul2[j2];
```

Tiling can help significantly

		Improvement
CPU_CLK_UNHALTED / INST_RETIRED	1.29	7.67%
RESOURCE_STALLS.RS_FULL / CPU_CLK_UNHALTED	8.87%	0.00%
IFU_MEM_STALL / CPU_CLK_UNHALTED	8.34%	7.02%
L1D_CACHE_LD / CPU_CLK_UNHALTED	0.23	11.72%
L1D_REPL / INST_RETIRED	1.32%	91.44%
L2_LINES_IN.ANY / INST_RETIRED	0.90%	31.84%
L1D_CACHE_LD / INST_RETIRED	4.58%	29.03%
L1D_CACHE_ST / INST_RETIRED	0.00%	99.44%

Where is Time Spent?

- It's simple if looking at the code is sufficient

```
    res[i][j] += mul1[i][k] * mul2[k][j];
```

- Use oprofile and observe location of events
- Select all interesting counters with opcontrol
- opannotate –source
  - Show all counters next to each line
- Opannotate –assembly
  - Show next to assembler instructions
  - Not precise since PEBS is not supported!

Annotated Listing

Function Total

```
11058   2.5025   111   0.5398   :{ /* lookup total: 162984 36.8848  8496 41.3129 */
                                :  unsigned long int hash;
                                :  size_t idx;
    2 4.5e-04      0        0   :  hash_entry *table = (hash_entry *) htab->table;
84957 19.2266   1693   8.2324   :  hash = 1 + hval % htab->size;
                                :  idx = hash;
11053   2.5014  1138   5.5337   :  if (table[idx].used) {
 6245   1.4133   340   1.6533   :      if (table[idx].used==hval && table[idx].keylen == keylen
                                :          && memcmp (table[idx].key, key, keylen) == 0)
                                :        return idx;
 2397   0.5425  1454   7.0703   :      hash = 1 + hval % (htab->size - 2);
                                :      do {
                                :          if (idx <= hash)
 3568   0.8075   292   1.4199   :            idx = htab->size + idx - hash;
                                :          else
                                :            idx -= hash;
30993   7.0140  2634  12.8082   :          if (table[idx].used==hval&&table[idx].keylen==keylen
```

L1 Cache Load

L2 Cache Load

Problems of Parallelism

- False sharing of cache lines:
  - Unintentionally use same cache line in different threads
  - Happens with global variables
  - Should not happen that often with dynamic memory
  - Group variables and align them
- Common working set:
  - Multiple threads working on same data (good!)
  - Produced output placed in same memory location (bad!)
  - Use per-thread working area and consolidate in end
- Synchronization:
  - Highly contested cache lines for sync primitives

Ratios for Multi-Thread Problems

- Modified Data Sharing Ratio:

  EXT_SNOOP/INST_RETIRED

  Instructions which cause modified cache line from other core to be retrieved

- Locked Operations Impact:
  (L1D_CACHE_LOCK_DURATION+20*L1D_CACHE_LOCK) / CPU_CLK_UNHALTED

  How many cycles used for atomic operations. Should be near zero

Summary

- There are many layers to memory performance
- Each program has different characteristics
- Statistical profiling can
  - Give general overview
  - Pinpoint hotspots
- Often program logic has to be significantly rethought

Questions?

Backup Slides

Common Memory Performance Problems

- Higher-level language semantics for strings
  - String implementation cannot guess what is needed
  - Programmer should provide estimates to C++ class
- In general: temporary values

C++ Temporaries

```
vec operator+(vec a, vec b) {
  vec r;
  for (size_t c = 0; c < a.size; ++c)
    r.data[c] = a.data[c] + b.data[c];
  return r;
}
```

Good style, bad performance

C++ Temporaries Help in C++0x

```
vec &&operator+(vec &&a, vec &&b) {
  for (size_t c = 0; c < a.size; ++c)
    a.data[c] += b.data[c];
  return a;
}
```

- && indicates rvalue reference, i.e., reference to temporary
- Can be reused for result instead of new allocation