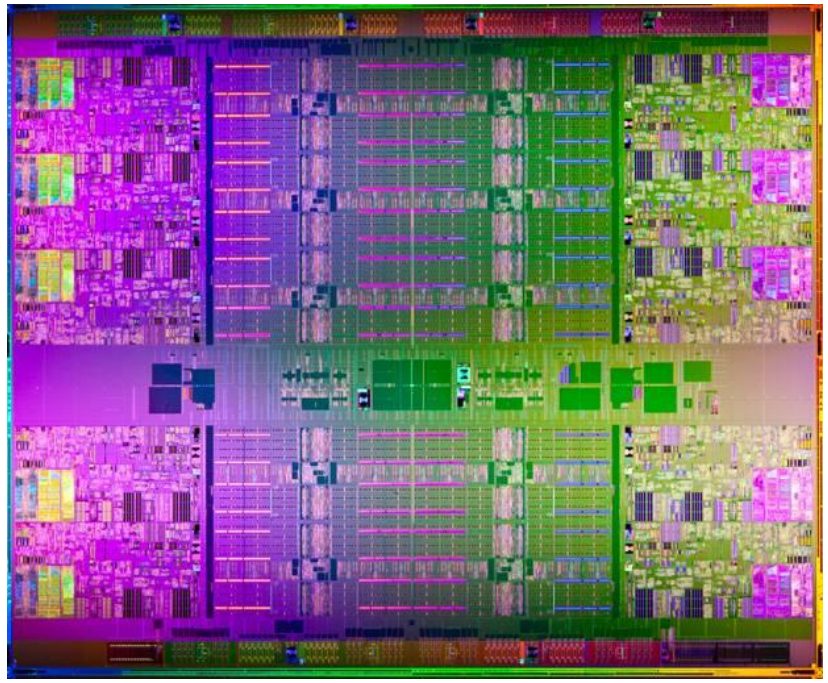
Optimizing memory bandwidth by exploring cache utilization efficiency at a sub-cacheline level

William Cohen
Performance Tools Engineer

Ben Woodard
Senior Principal Consultant

Registers & Cache



Registers - as fast as processors

- Lots of work goes into register optimization within compilers
- Variables move over course of function - location lists
- ABI adds constraints

Cache - still very fast.

- Optimization done almost exclusively in the source not by the compiler
- Always transferred in lines - often 64 bytes
- Cost in die size & power
- Very limited resource

Memory

Memory bandwidth hamstrings processor's capability

Bandwidth impact of selecting address.

- Streaming through columns the fastest
- Impact of changing rows - vector operations

all those maps

```
#pragma omp target teams map (alloc: a, b, c, d, u, indxp, jndxp, rho_i, qs) \
num_teams((npl+127)/128)
```

```
#pragma acc parallel num_gangs(n3-2) num_workers(8) vector_length(128) #pragma omp target
map(tofrom: ou[0:n3*n2*n1]) map(tofrom: ov[0:n3*n2*n1]) map(tofrom: or[0:n3*n2*n1])
is_device_ptr(u1, u2)
```

Not just moving data

Restructuring data

Linearizing

Optimizing stride

Memory & Cache

The problem with C/C++ structs & classes

Each cache miss transfers a full 64B cacheline. How much of that data is being used?

- A hot cacheline may only have 8B being used.
- (the other 56B could be wasted bandwidth)

Read to write

Pointers indirection (pointer following)

Structure of arrays vs. arrays of structures

- The problem with STL (and even PSTL) containers and algorithms.
- Programmer conceptual organization vs optimal memory access

Strides

Quest: Mem bandwidth efficiency

How much of the cache is actually being used? How much memory bandwidth is being used to transfer useful data?

Cache is SRAM very power hungry (vs DRAM)

Power cost of moving bits

Increase efficiency :

- Increase speed - processor waits less often
- Reduce power
- More effective memory bandwidth

First attempts:

No performance counters - Cache miss an approximation but not a good one.

    Only a few bytes in within a cache line

Pahole - analyzes data structures using DWARF based on cachelines

Valgrind’s cachegrind, HPC Toolkit to identify target regions

    Data attribution problem

Rosetta Stone (Debug Info)

Debug info provides:

- Mapping between source code lines and machine instructions
- Information for unwinding the stack to get backtraces
- Location information for variables

Physical layout of various data structures

- Due to its utility by default Red Hat generates optimized debug info and extracts it into debuginfo RPM
- (Leave optimization on, GCC has better quality of debuginfo for optimized code)

Static Analysis of data structs

Looking for bytes in data structures that force alignment but hold no useful data

The padding bytes will consume memory bandwidth as data moved between RAM and CPU

The dwarves pahole tool:

- Uses the debuginfo to determine struct layout
- Developed by linux kernel engineers to examine data structure layout
  - Find related struct fields in different cache lines
  - Find unused regions in data structures

Pahole Example from lulesh2.0

```
class Domain {

public:
    void Domain(class Domain *,Int_t,Index_t,Index_t,Index_t,Index_t,Int_t,Int_t,Int_t,Int_t);
    void ~Domain(class Domain *, int);
    ...
    real_t *                    commDataSend;      /*  0   8 */
    ...
    /* --- cacheline 7 boundary (448 bytes) was 8 bytes ago --- */
    class vector<double, std::allocator<double> > m_y; /*   456     24 */
    class vector<double, std::allocator<double> > m_z; /*   480     24 */
    ...
```

Pahole Example (continued)

```
    Int_t                         m_cycle;              /*  1592     4 */
    /* XXX 4 bytes hole, try to pack */
    /* --- cacheline 25 boundary (1600 bytes) --- */
    Real_t                        m_dtfixed;            /*  1600     8 */
    Index_t                       m_planeMax;           /*  1740     4 */
    ...
    /* size: 1744, cachelines: 28, members: 104 */
    /* sum members: 1740, holes: 1, sum holes: 4 */
    /* last cacheline: 16 bytes */
};
```

Drawbacks of pahole analysis

Static analysis:

- No information about actual amount of wasted memory/bandwidth
- No information about which members have temporal locality

Explicitly named fields assumed accessed:

- Wouldn't catch unused element in float4 used to store 3d info (cuda particles example)

Only examines structs, no analysis of adjacent variables or array element access

Assumes struct starts at beginning of cacheline (might not be right for malloced or stack locals)

Intentional padding might improve performance (for example unrelated locks in separate cache lines)

Dynamic Analysis

Valgrind framework has instrumentation for:

- Memory allocation, heap profiling, thread debugging, and cache simulation

Callgrind tool “spatial loss” metric (Weidenforfer and Breitbart 2016)

- Cache simulator tracks cache line byte accesses
  - When cache line evicted note number of bytes unaccessed in cache line
  - Attribute unused bytes to the location where cache line initially loaded (not where evicted)
- Kcachegrind tool reads callgrind data and maps data back to lines in source code

Cache efficiency and Spatial loss

Get an estimate of how much of the cache space is unused (wasted):

Waste = (spatial_loss)/(bytes_per_cache_line*cache_misses)

Example:

Sploss1: 23,312,824,492 bytes

Cache line: 64 bytes

Cache misses: I1mr: 791,680 D1mr: 2,538,385,921 D1mw: 949,383,013

Cache misses total: 2,634,115,614

23,312,824,492 bytes/(64 bytes*2,634,115,614) = 13.8% waste

Callgrind Disadvantages

Very slow:

- Instrumentation makes thread >100x slower than native
- Valgrind serializes threads, so native 8 threads in parallel another 8x slower

Spatial loss metric missing information to compute wasted memory read/write bandwidth

Kcachegrind data presentation focuses where in the code rather where in the data

Cache bytes Use-Def tracking

Want to split callgrind access tracking into:

- Use bytes (bytes in cache where the value is read from memory)
- Def bytes (bytes in cache where the value is written to memory)

A read after a write (Def) does not count as a Use , the value not read from memory

Compute bandwidth waste in manner similar to spatial loss for cache line:

- Read (use_byte>0) ? (line_size_bytes - use_bytes) : 0
- Write (def_byte>0) ? (line_size_bytes - def_bytes) : 0

Memory Bandwidth Calculations

Wasted read bandwidth:

Sum(Use_bytes)/(cache_line_bytes * loaded_lines)

Wasted write bandwidth:

Sum(Def_bytes)/(cache_lines_bytes * modified_lines_evicted)

Mapping Info to Data structs

Kcachegrind maps information to lines in source code

Which variable in the following complex expression is the problem? hxx? hourgam? zd?

```
hxx[i] = hourgam[0][i] * zd[0] + hourgam[1][i] * zd[1] +
         hourgam[2][i] * zd[2] + hourgam[3][i] * zd[3] +
         hourgam[4][i] * zd[4] + hourgam[5][i] * zd[5] +
         hourgam[6][i] * zd[6] + hourgam[7][i] * zd[7];
```

Want to map the data to the variable/field accessed

Make use of the debuginfo column information in generated by default in GCC 8

Inverted location lists

DWARF designed for source debuggers: map f(variable name, IP) -> location of variable

Need: instruction i.e. IP -> operands (memory or registers) -> variable being addresses

Compiler has info but not emitted ”Too big” “not needed“

DWARF it ain’t just for GDB anymore

Several uses: Cache Locality and efficiency, data attribution

Challenges: arrays and indexing, interprocedure, compiler optimizations

Future Work

Implement def-use cache line tracking in Valgrind

Implement debuginfo column use in Valgrind/kachegrind

Explore better alternatives to map data back to variables and struct fields

Techniques for cache allocation similar to traditional compiler register allocation

Further Information

Dwarf debuginfo - http://dwarfstd.org/

Pahole tool - https://github.com/acmel/dwarves

Valgrind/Callgrind

http://www.valgrind.org/

Inclusive Cost Attribution for Cache Use Profiling (Weidendorfer and Breitbart 2016)