HOW FAST CAN YOU GO – OPTIMIZING MEMORY CACHE PERFORMANCE

CLAIRE CATES
DISTINGUISHED DEVELOPER
CLAIRE.CATES@SAS.COM

AGENDA

- Terms
- Performance Problems
- Tools I’ve used

MEMORY TERMS

Latency

- The delay to access the memory.
- Usually measured in clock cycles to return the requested data.
- The slower the latency, the slower your program runs.

Bandwidth

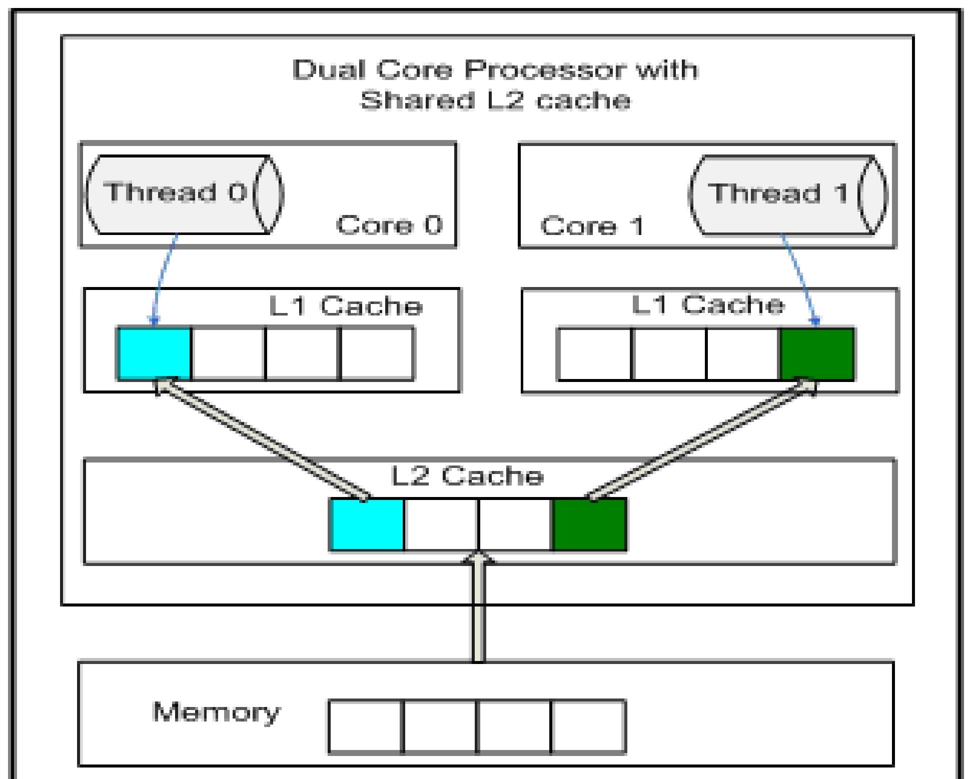
- The pipeline carrying the memory from main memory to the processor.
- If you saturate the pipeline, performance will be impacted

MEMORY CACHE

- Used by the CPU to reduce the memory latency
- A section of Memory closer to the CPU
- Stores frequently used memory
- Design assumptions for the cache.
  - Data that is accessed once will more than likely be accessed again
  - When memory is accessed, memory near that location will be accessed.

MEMORY CACHE

- Instruction Cache – used for executable instructions
- Data cache – used to speed up data fetch and store
  - L1 (Level 1) – closest cache to the CPU – fastest – smaller
  - L2 (level 2) – if data is not in the L2 cache – slower than L1 but faster than main memory, larger than L2.
  - L1 – L2 … caches may be shared on multi-core systems
  - Many systems now have an L3 cache



```
> lscpu
Architecture:           x86_64
CPU op-mode(s):         32-bit, 64-bit
Byte Order:             Little Endian
CPU(s):                 16
On-line CPU(s) list:    0-15
Thread(s) per core:     2
Core(s) per socket:     4
CPU socket(s):          2
NUMA node(s):           2
Vendor ID:              GenuineIntel
CPU family:             6
Model:                  44
Stepping:               2
CPU MHz:                3059.050
BogoMIPS:               6117.78
Virtualization:         VT-x
L1d cache:              32K
L1i cache:              32K
L2 cache:               256K
L3 cache:               12288K
NUMA node0 CPU(s):      0,2,4,6,8,10,12,14
NUMA node1 CPU(s):      1,3,5,7,9,11,13,15
```

MEMORY CACHE TERMS

Cache Line

Data is copied from main memory in a fixed size area. Typically 64 bytes long. Cache lines will be copied from main memory to satisfy the data request. Multiple cache lines may be copied.

Cache Hit

The data is found in the cache

Cache Miss

The data is not found in the cache. The CPU will need to load it from a higher level cache or main memory. You want to avoid Cache Misses.

MEMORY CACHE TERMS

Dirty Cache Line

When data is written to memory it needs to eventually be written back to main memory. It is dirty, if the contents have not been written back.

Write-back policy

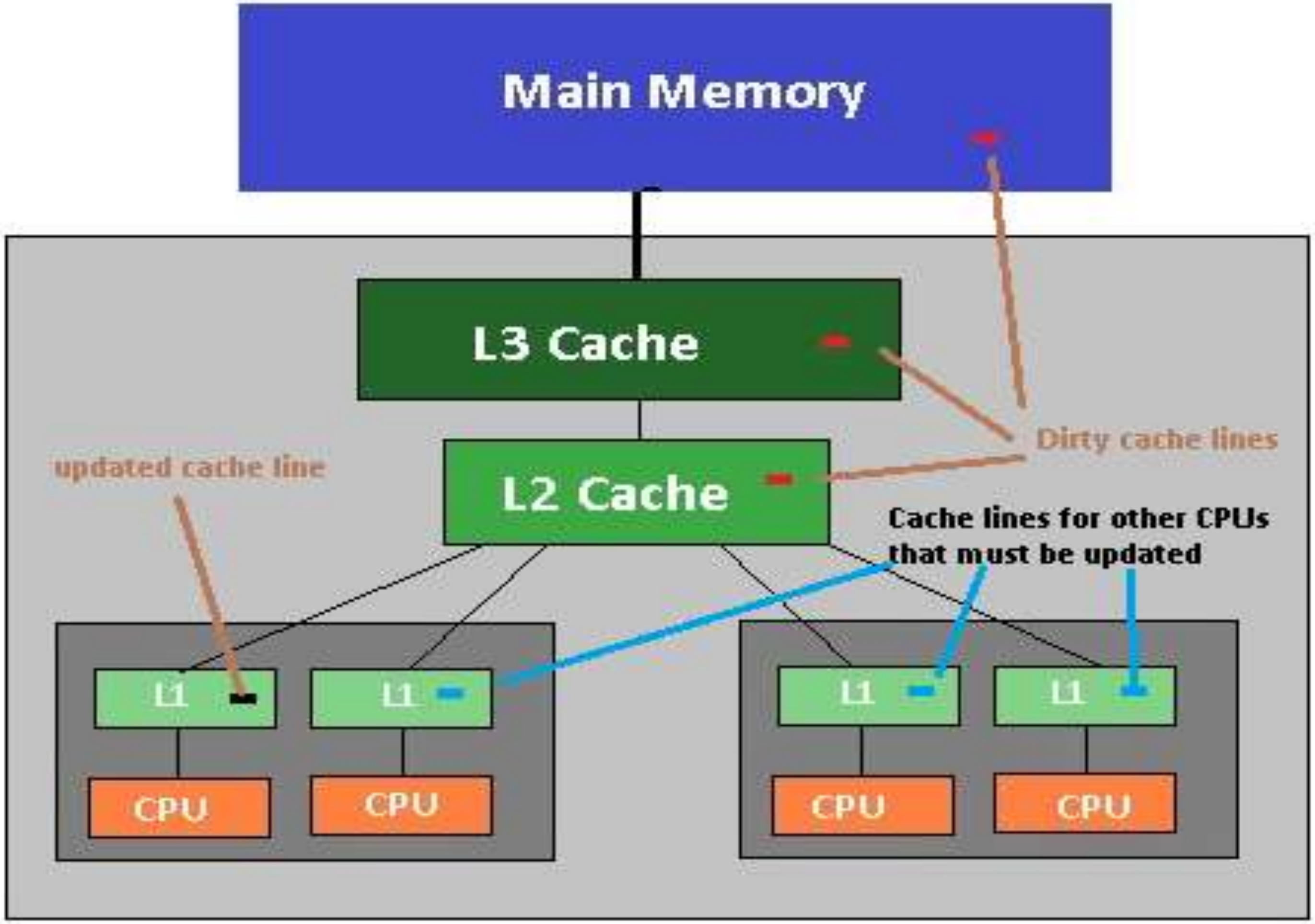
The policy the CPU uses to determine when to write the dirty cache lines back to main memory.

Cache Coherence

Multiple CPU caches have a private copy of the same piece of memory.

The process of making sure each of these copies have the updated “correct” content.

DIRTY CACHE LINE AND CACHE COHERENCY



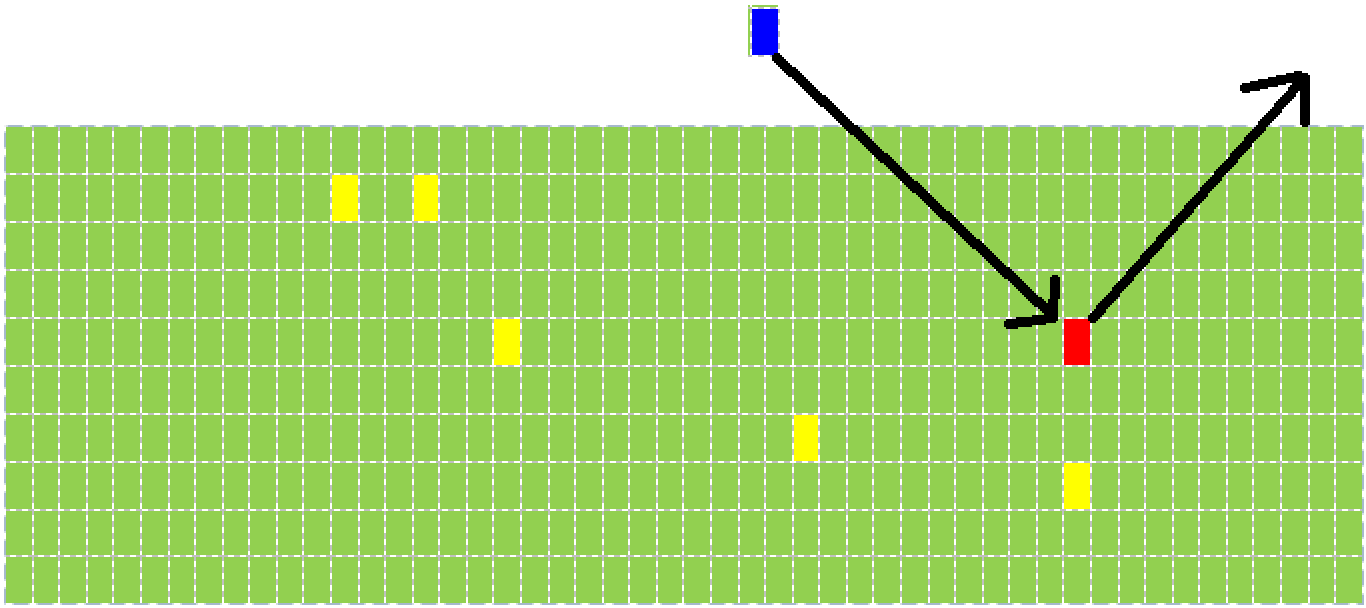
MEMORY CACHE TERMS

Evicted

As the cache becomes full and more cache lines are loaded, an existing cache line will need to be evicted.

Replacement Policy

The policy the CPU uses to determine which cache line to evict. LRU is the most commonly used policy.



MEMORY CACHE TERMS

Memory is allocated in pages

Pages

A fixed size (**page size**) block of memory that is mapped to areas of physical memory. Page Size is often 4K

Page Table

The page table contains the translation from the virtual address to the physical address for the pages.

MEMORY CACHE TERMS

Applications access memory virtually

Translation Lookaside Buffer (TLB)

- Used to speed up Virtual to Physical address translation.
- TLB contains the recent mapping from the page table.

Prefetching

The CPU guesses at what memory will be needed next and loads it.

- Guess right can save latency
- Guess wrong, can cost bandwidth and cache line evictions.

MEMORY CACHE PERFORMANCE ISSUES

- Performance problems occur when there are a lot of cache misses.
- Best to look at the ratio of cache misses to cache hits.
- Accessing memory that is in the lower level caches is the best
- Accessing memory sequentially is the best – prefetching
- Full Random is the worst – prefetching is loading bad data and TLB misses.
- Cache misses may also cause further delay if the bandwidth become saturated.

MEMORY CACHE PERFORMANCE ISSUES

Performance Implications of a Cache Miss

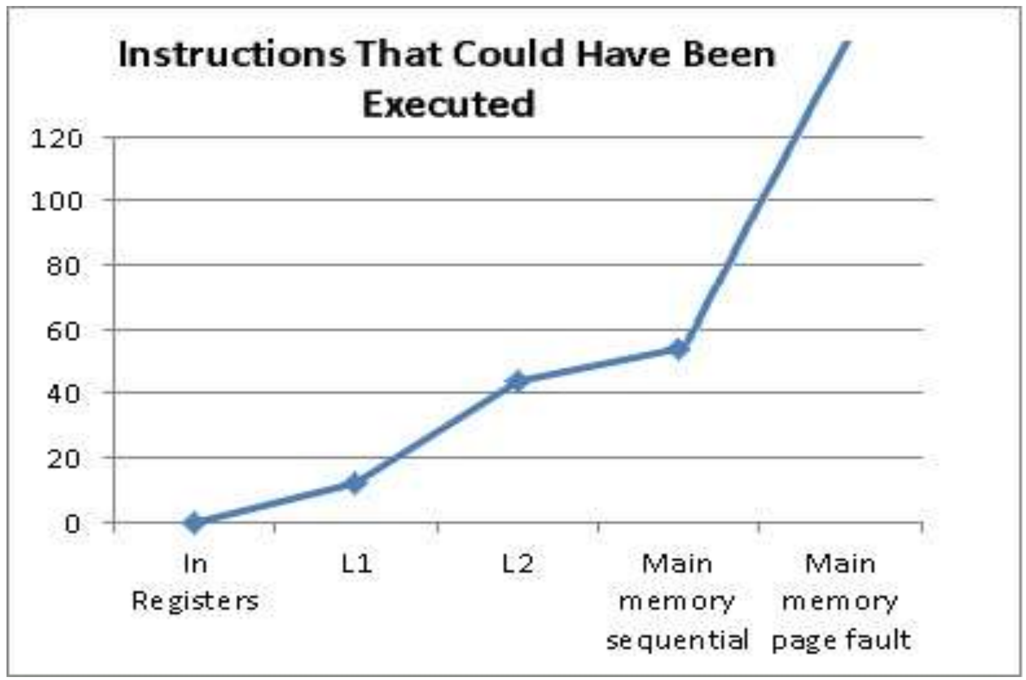
	L1 Cache	L2 Cache	L3 Cache	Main memory
Sequential	4 clk	11 clk	14 clk	6ns
In-Page Random	4 clk	11 clk	18 clk	22ns
Full Random	4 clk	11 clk	38 clk	65.8 ns

Sandy Bridge Latencies for accessing memory. Clk stands for clock cycles and ns stands for nanoseconds.

HOW MUCH CAN LATENCY REALLY AFFECT THE SYSTEM?

From the SandyBridge numbers.

- Assume 3GHz processor executes 3 instructions per cycle
- Going to the L1 cache the processor stalls for 4 clk or the CPU could have executed 12 instructions.
- If the memory is in the L2 cache the CPU could have executed 44 instructions.
- Sequentially accessing main memory would result in stalling the CPU for 6*9 (54) instructions.
- Randomly accessing main memory could result in stalling the CPU for almost 600 instructions.

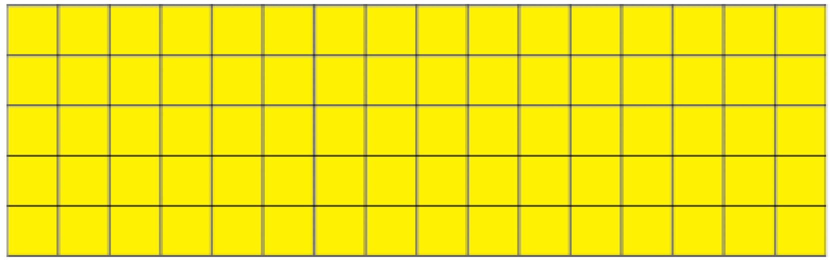


Cache Problems

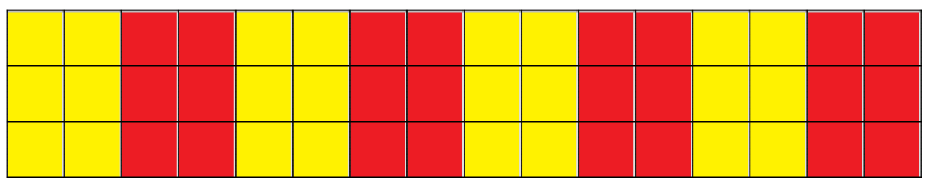
Remember those Design assumptions for the cache

- Data that is accessed once will more than likely be accessed again
- When memory is accessed, memory near that location will be accessed.

DATA LAYOUT CACHE PROBLEM (FETCH UTILIZATION)



Good Utilization all memory in the cache is used



Poor utilization, only half of the memory in the cache is used. The other memory takes up cache space and also needs to be moved thru the pipeline.

FETCH UTILIZATION

- In your structures,
  - Put data that is used often together
    - So that the used data is all in the cache and rarely used data is not loaded into the cache

  - If needed break up your into multiple structures. This is especially important if the structures are in an array.

```
struct Good {
  int   used;
  int   used2;
}
struct Good2 {
  int    not_used;
  int    not_used2;
}
```

```
struct Bad {
      int used;
      int used2;
      int not_used;
      int not_used2;
}
```

FETCH UTILIZATION

- Put data that is written together
  - Data that changes may affect other cache lines – reduce the number of writebacks – especially in data that is shared across threads.

- Make sure data items are sized correctly.
  - If you only need a short, don't use an int or long. The extra bytes are wasted.

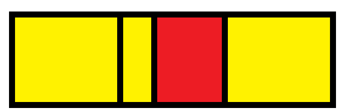
- Using small memory allocations can be very wasteful
  - Causes a random pattern
  - Often times memory allocators allocate more than the real size for headers,….

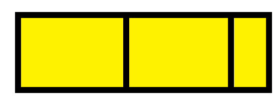
FETCH UTILIZATION

- Account for the alignment of the data items.
  - Keep data items in a structure that have similar sizes near each other

Struct Bad {
int a;
char b;
int c;
}

Struct Better {
int a;
int c;
char b;
}



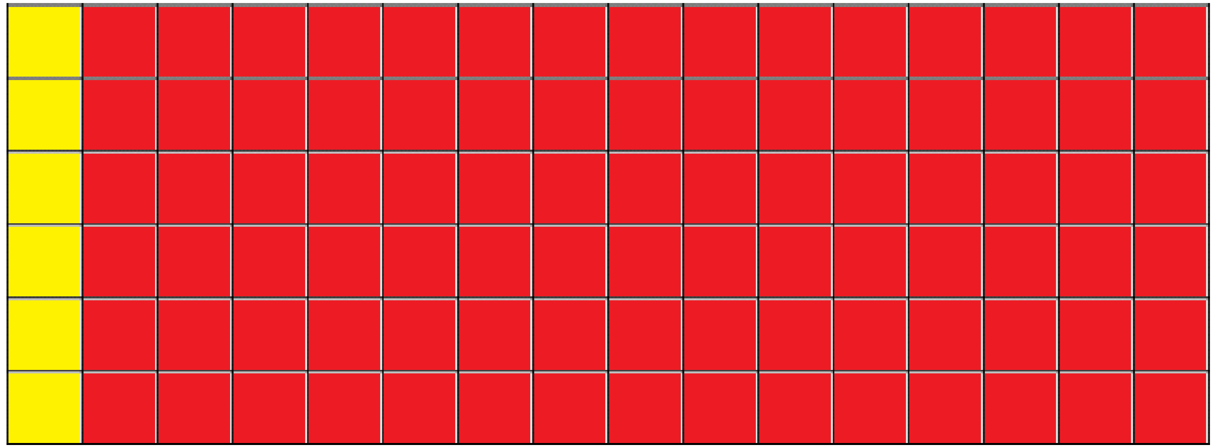


char	1 byte	1 byte aligned
Short	2 bytes	2 byte aligned
Int / long	4 bytes	4 byte aligned
Float	4 bytes	4 byte aligned
Double (Windows)	8 bytes	8 byte aligned
Double (Linux)	8 bytes	4 byte aligned
Long double	8-12 bytes	4 – 8 byte aligned

Data Access Problems

Once the data is in the cache, use the cache line as much as possible before it is evicted!

DATA ACCESS – NON-TEMPORAL DATA



```
for( i=0; i<num_cols;i++)
    for( j=0; j< num_rows; j++ )
        do something with the array element
```

1. Accessing in Row order would use all the memory in the cache. Accessing in column order runs out of cache before the memory can be reused.
2. Access the memory sequentially for prefetch gains.
3. Non-Temporal access pattern can occur if you are just trying to analyze too much memory at once even if it is not in a loop. Break it up into smaller chunks and combine at the end if possible.

DATA ACCESS – NON-TEMPORAL DATA

```
for( i=0; i<10;i++)
    for( j=0; j< bigsize; j++ )
        mem[j]  += cnt[j] + arr[i];
```

- If bigsize is large enough, the code will execute and load each cache line into memory but the cache line will be evicted before the next iteration.

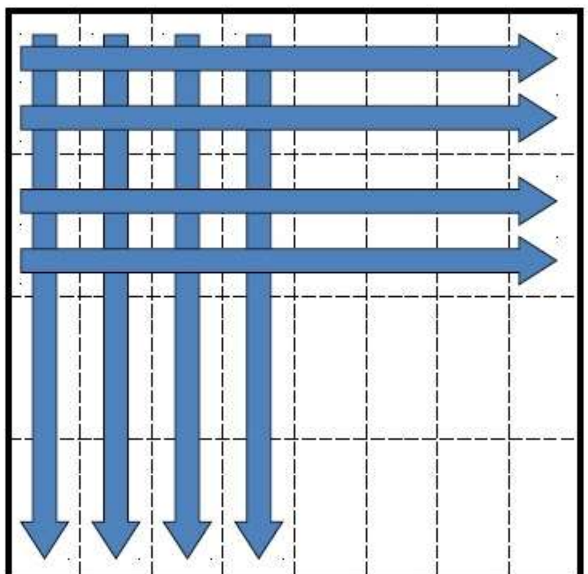
```
for( i=0; i<bigsize;i++)
    for( j=0; j< 10; j++ )
        mem[i]  += cnt[i] + arr[j];
```

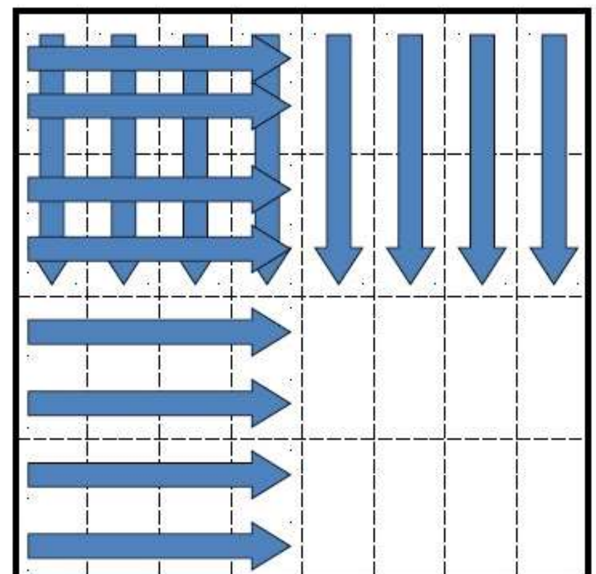
- This will keep the cache line in memory for the full duration of the loop of 10 where it is used.

DATA ACCESS – NON-TEMPORAL DATA

BREAK THE DATA BEING PROCESSED UP INTO SMALLER BLOCKS

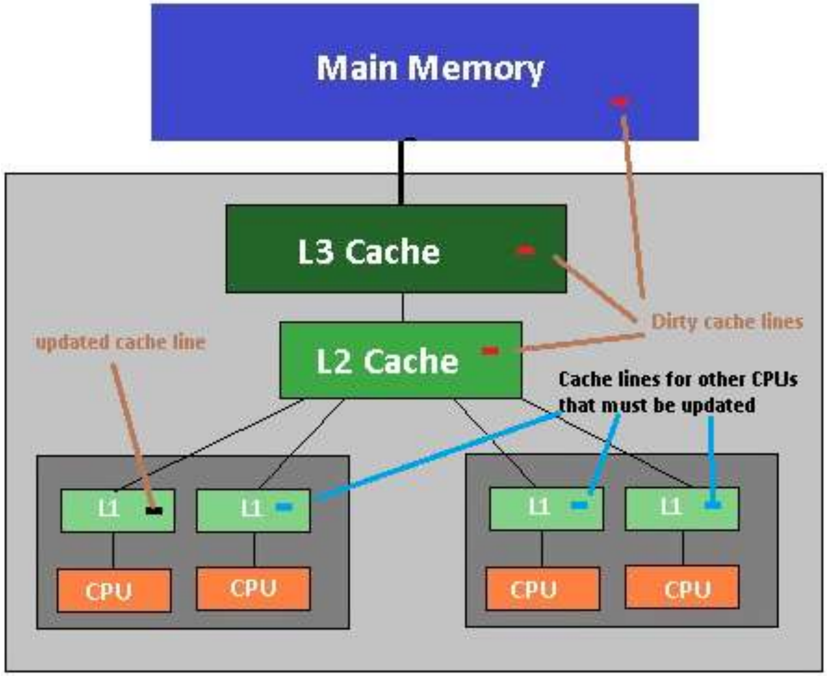
```
double M1[cnt][cnt], M2[cnt][cnt], alpha;
for (i = 0; i < cnt; i++)
    for (j = 0; j < cnt; j++)
        M1[i][j] += M2[j][i] * alpha;

for (ii = 0; ii < cnt; ii += 8)
    for (jj = 0; jj < cnt; jj += 8)
        for (i = ii; i < ii + 8; i++)
            for (j = jj; j < jj + 8; j++)
                M1[i][j] += M2[j][i] * alpha;
```





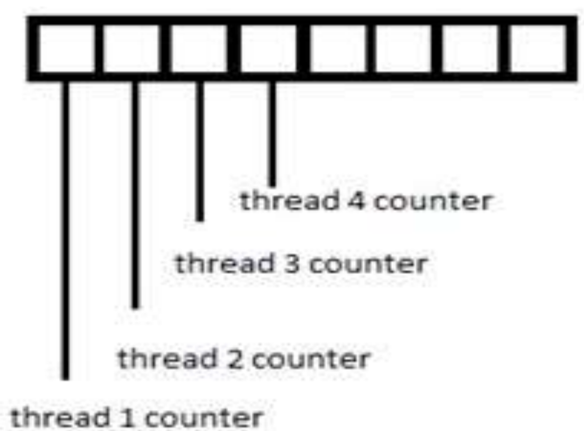
CACHE COHERENCY AND COMMUNICATION UTILIZATION

- When 2 or more threads share a common memory area and any data is written, cache problems can occur.
- When one thread writes to the area the cache for the other thread(s) will be invalidated.
- Care should be taken to reduce the number of writes into shared memory.



FALSE SHARING

- 2 or more threads are using data in the same cache line.
- 1 thread writes to the cache line and it invalidates the data in the other thread(s) cache line
- Often seen when allocating arrays of data based on the number of threads and shared by the threads



- Avoid false sharing by placing data that can change, close together. Reading data does not destroy the cache.
- Align memory on a cache line boundary. (pad structures if necessary)

RANDOM MEMORY ACCESS

- Caches work best when memory that is near an already loaded cache line is accessed.
- Memory allocations produce random access to memory.
- Random access patterns can cause TLB misses which can be costly
- Linked list, hashes, tree traversals can also produce a random access memory pattern.

TOOLS

- Amplifier – with general exploration will tell you some information about the performance. Several of the counters deal with the cache. The tool will point you to the code and assembler code that is causing the problems.

- ThreadSpotter –It is solely looking at memory usage and will show you the areas in your program where the cache is not utilized thoroughly, where sharing between threads is hurting the case, false sharing and loop order issues.   Gives source code and a good description of the issues involved.

- I use both tools to get a better idea of where we are spending performance cycles.

ThreadSpotter: sas (12M/64)

ThreadSpotter™

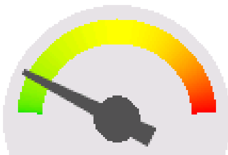
ThreadSpotter™ is a tool to quickly analyze an application for a range of performance problems, particularly related to multicore optimization.

Read more... Manual

Open the Report

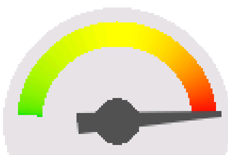
Your application

Application: /sasgen/dev/mva-v940m1/SAS/laxnd/sas iceland -verify_paths -set SASROOT /sasgen/dev/mva-v940m1/SAS/laxnd -widebug noimgunload -set tkopt noext_unload opt -notkmmemfill -notkmosfill -memsize 0 -config /sasgen/dev/mva-v940m1/SAS/laxnd/sasv9.cfg -config /sasgen/dev/mva-v940m1/SAS/laxnd/nls/en/sasv9.cfg -helphost d77358.na.sas.com



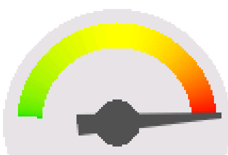
Memory Bandwidth

The memory bus transports data between the main memory and the processor. The capacity of the memory bus is limited. Abuse of this resource limits application scalability.
Manual: Bandwidth



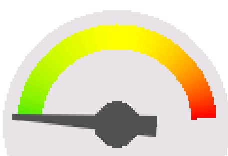
Memory Latency

The regularity of the application's memory accesses affects the efficiency of the hardware prefetcher. Irregular accesses causes cache misses, which forces the processor to wait a lot for data to arrive.
Manual: Cache misses Manual: Prefetching



Data Locality

Failure to pay attention to data locality has several negative effects. Caches will be filled with unused data, and the memory bandwidth will waste transporting unused data.
Manual: Locality



Thread Communication / Interaction

Several threads contending over ownership of data in their respective caches causes the different processor cores to stall.
Manual: Multithreading

This means that your application shows opportunities to:

Tune cache utilization to avoid processor stalls.

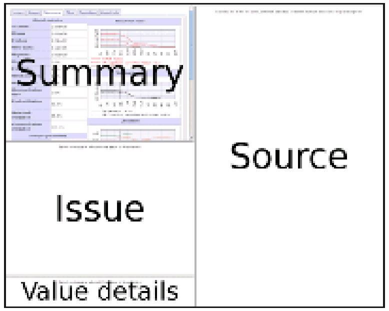
Read more...

Next Steps

The prepared report is divided into sections.

- Select the tab **Summary** to see global statistics for the entire application.
- Select the tabs **Bandwidth Issues**, **Latency Issues** and **MT Issues** to browse through the detected problems.
- Select the tab **Loops** to browse through statistics and detected problems loop by loop.

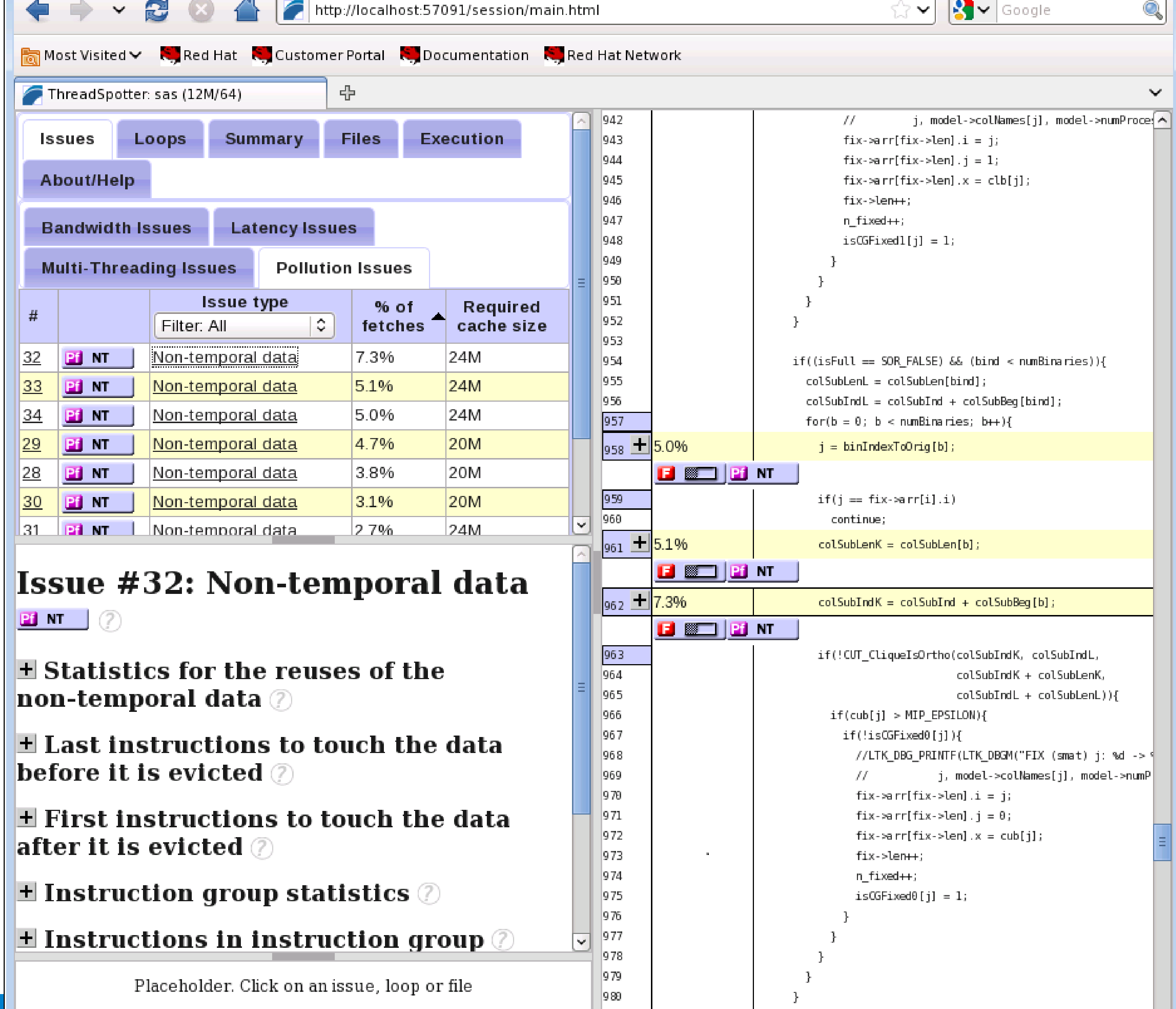
The Issue and Source windows contain details and annotated source code for the detected problems.



Resources

Manual

Table of Contents Overview
Optimization Workflow Concepts
Reading the Report Issue Reference


http://localhost:57091/session/main.html
Most Visited
Red Hat
Customer Portal
Documentation
Red Hat Network
ThreadSpotter: sas (12M/64)
Issues
Loops
Summary
Files
Execution
About/Help
Bandwidth Issues
Latency Issues
Multi-Threading Issues
Pollution Issues
| Issue type | % of fetches | Required cache size
Filter: All
32 Pf NT Non-temporal data 7.3% 24M
33 Pf NT Non-temporal data 5.1% 24M
34 Pf NT Non-temporal data 5.0% 24M
29 Pf NT Non-temporal data 4.7% 20M
28 Pf NT Non-temporal data 3.8% 20M
30 Pf NT Non-temporal data 3.1% 20M
31 Pf NT Non-temporal data 2.7% 24M
Issue #32: Non-temporal data
Statistics for the reuses of the non-temporal data
Last instructions to touch the data before it is evicted
First instructions to touch the data after it is evicted
Instruction group statistics
Instructions in instruction group
Placeholder. Click on an issue, loop or file
fix->arr[fix->len].i = j;
fix->arr[fix->len].j = 1;
fix->arr[fix->len].x = clb[j];
fix->len++;
n_fixed++;
isCGFixed1[j] = 1;
if((isFull == SOR_FALSE) && (bind < numBinaries)){
colSubLenL = colSubLen[bind];
colSubIndL = colSubInd + colSubBeg[bind];
for(b = 0; b < numBinaries; b++){
5.0% j = binIndexToOrig[b];
if(j == fix->arr[i].i)
continue;
5.1% colSubLenK = colSubLen[b];
7.3% colSubIndK = colSubInd + colSubBeg[b];
if(!CUT_CliqueIsOrtho(colSubIndK, colSubIndL,
colSubIndK + colSubLenK,
colSubIndL + colSubLenL)){
if(cub[j] > MIP_EPSILON){
if(!isCGFixed0[j]){
fix->arr[fix->len].i = j;
fix->arr[fix->len].j = 0;
fix->arr[fix->len].x = cub[j];
fix->len++;
n_fixed++;
isCGFixed0[j] = 1;

8.10. Non-Temporal Data - Mozilla Firefox

http://localhost:42992/doc/manual_html/r

8.10. Non-Temporal Data

8.10. Non-Temporal Data

A *non-temporal data* issue is reported when ThreadSpotter™ finds places where accessed cache lines are nearly always evicted from the cache before being reused. However, the cache lines still occupy space in the cache, that could otherwise be put to better use. See Section 5.3, "Non-Temporal Data" for more information about non-temporal data.

Using non-temporal prefetches on the the non-temporal data can prevent the data from being cached in this cache level. This does not hurt performance since the data would have been evicted from the cache before being reused anyway, but may improve performance by leaving more cache space for other data that can be successfully cached, and for data of other threads and processes that are sharing the cache. See Section 5.3.5.1, "Non-Temporal Prefetches" for more information.

This issue type is normally only included when analyzing the highest cache level, that is, the cache level closest to memory, since non-temporal prefetches affect this cache level in most processors.

Issue #12: Non-temporal data Pf NT

Statistics for the reuses of the non-temporal data

% fo fetches	6.5%
Fetch ratio	100.0%
Fetches	8.08e+05

Last instructions to touch the data before it is evicted

Stack	Instruction	% of non-temporal reuses	Required cache size

<no current project> - Intel VTune Amplifier

File View Help

Welcome | r002ge | r003ge

General Exploration General Exploration viewpoint (change) ?

Intel VTune Amplifier XE 20

Analysis Target | Analysis Type | Summary | Bottom-up | Top-down Tree | Tasks and Frames

Instructions Retired: 54.000.000

Filled Pipeline Slots

%ErrorInHeaderCalculation

%ErrorInHeaderCalculation

Unfilled Pipeline Slots (Stalls)

Back-end Bound: 0.872

Memory Latency

LLC Load Misses Serviced By Remote DRAM: 1.000

A significant amount of time is spent servicing memory requests from remote DRAM. Wherever possible, try to consistently use data on the same core, or at least the s
was allocated on.

LLC Miss: 0.000

LLC Hit: 0.000

DTLB Overhead: 0.204

A significant proportion of cycles is being spent handling first-level data TLB misses. As with ordinary data caching, focus on improving data locality and reducing working
DTLB overhead. Additionally, consider using profile-guided optimization (PGO) to collocate frequently-used data on the same page. Try using larger page sizes for large

Contested Accesses: 0.000

Data Sharing: 0.000

Memory Reissues

Front-end Bound: 0.128

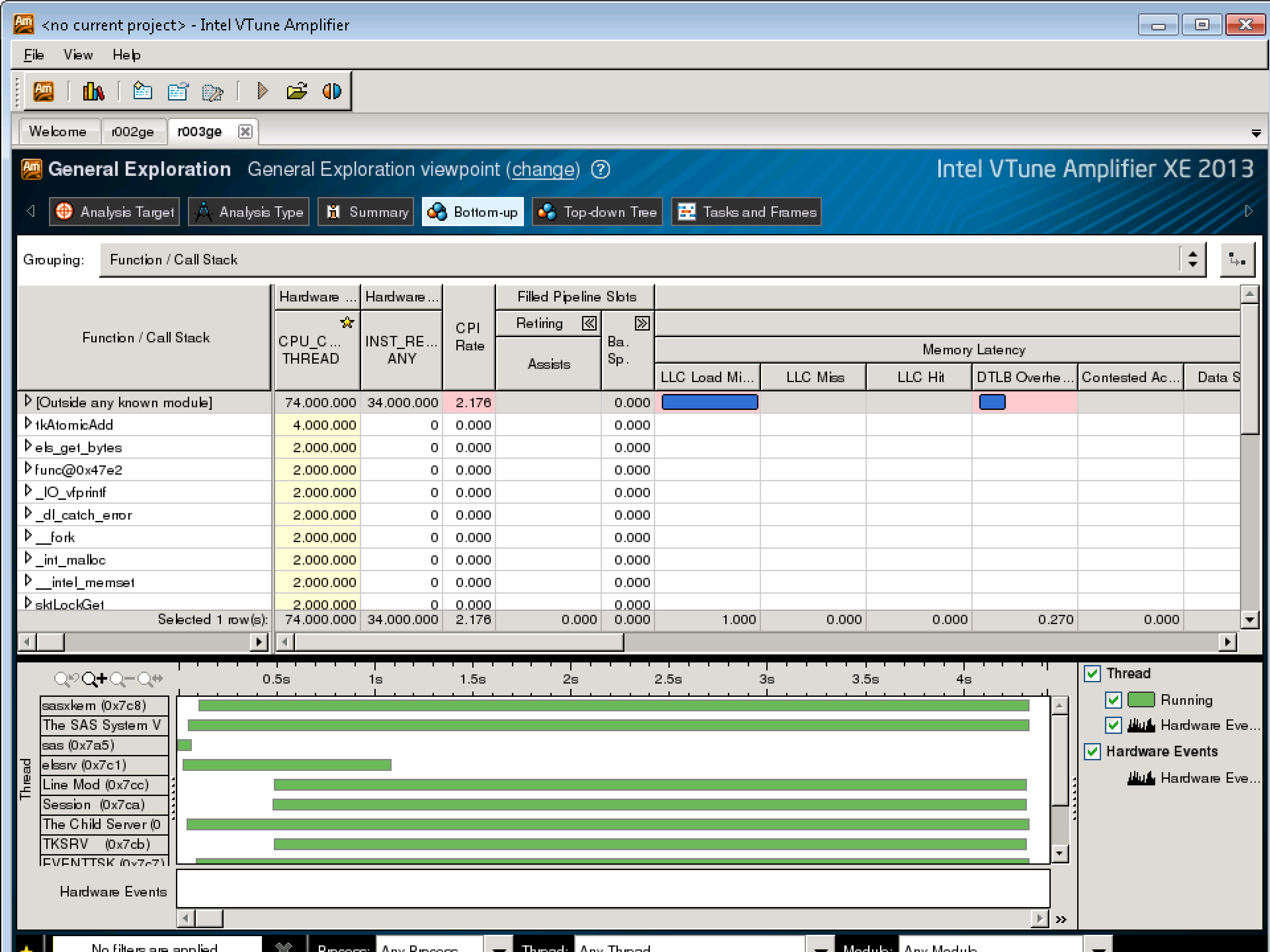
ICache Misses: 0.019

A significant proportion of instruction fetches are missing in the instruction cache. Use profile-guided optimization to reduce the size of hot code regions. Consider compiler opti
functions so that hot functions are located together. If your application makes significant use of macros, try to reduce this by either converting the relevant macros to functions o

ITLB Overhead: 0.000

DSB Switches: 0.000

Collection and Platform Info


<no current project> - Intel VTune Amplifier
File View Help
Welcome
r002ge
r003ge
General Exploration
General Exploration viewpoint (change)
Intel VTune Amplifier XE 2013
Analysis Target
Analysis Type
Summary
Bottom-up
Top-down Tree
Tasks and Frames
Grouping:
Function / Call Stack
Function / Call Stack
Hardware ...
CPU_C ... THREAD
Hardware ...
INST_RE ... ANY
CPI Rate
Filled Pipeline Slots
Retiring
Assists
Ba. Sp.
Memory Latency
LLC Load Mi...
LLC Miss
LLC Hit
DTLB Overhe...
Contested Ac...
Data S
[Outside any known module] 74.000.000 34.000.000 2.176 0.000
tkAtomicAdd 4.000.000 0 0.000 0.000
els_get_bytes 2.000.000 0 0.000 0.000
func@0x47e2 2.000.000 0 0.000 0.000
_IO_vfprintf 2.000.000 0 0.000 0.000
_dl_catch_error 2.000.000 0 0.000 0.000
__fork 2.000.000 0 0.000 0.000
_int_malloc 2.000.000 0 0.000 0.000
__intel_memset 2.000.000 0 0.000 0.000
sktLockGet 2.000.000 0 0.000 0.000
Selected 1 row(s): 74.000.000 34.000.000 2.176 0.000 0.000 1.000 0.000 0.000 0.270 0.000
0.5s 1s 1.5s 2s 2.5s 3s 3.5s 4s
Thread
sasxkern (0x7c8)
The SAS System V
sas (0x7a5)
elssrv (0x7c1)
Line Mod (0x7cc)
Session (0x7ca)
The Child Server (0
TKSRV (0x7cb)
Hardware Events
Thread
Running
Hardware Eve...
Hardware Events
Hardware Eve...
No filters are applied
Process: Any Process
Thread: Any Thread
Module: Any Module

SUMMARY

- Cache's were designed with the assumption that
  - once memory is loaded it will likely be accessed again.
  - Memory that is accessed is likely close to other memory that will be used.

- Memory caches have improved performance but if a developer doesn't understand the principals of the cache and doesn't design with caches in mind, their application will suffer performance problems.

- Remember each time the CPU has to go back to main memory, the CPU will be stalled and not performing useful work.

- Not all issues that may be discovered will be fixable.

QUESTIONS

CLAIRE.CATES@SAS.COM