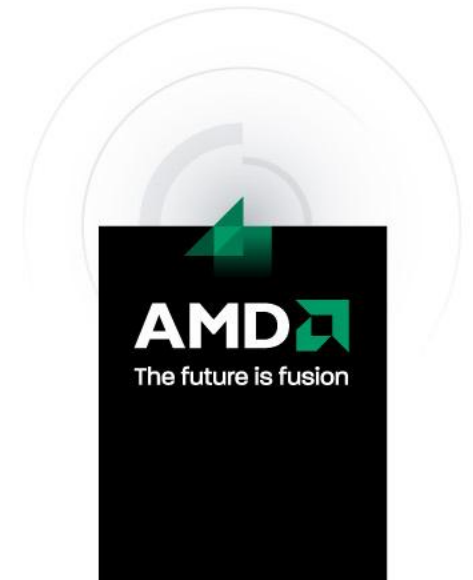


Cache Line Utilization with AMD CodeAnalyst Software

Toward Data-Centric Performance Analysis

Scott Gibbons, December 14, 2011



Agenda

- AMD CodeAnalyst Software Overview
- Cache Line Utilization (CLU) Introduction
- Cache Organization
- CLU Details in AMD CodeAnalyst Software
- Example
- Future Plans
- Conclusion



AMD CodeAnalyst Software Overview

- General Features:

- System-wide sampling-based performance analysis
- Low overhead
- Supports both Windows and Linux
- Performance analysis on managed and unmanaged code
- Easy to use Graphic Interface
- Command line utilities
- API libraries: data collection, data access, JIT notification



AMD CodeAnalyst Software Overview

- Future directions
 - Adding data-centric profiling features
 - Cache Line Utilization
 - Data False Sharing
 - Etc.
 - Adding more “analysis”
 - Hints on optimization opportunities

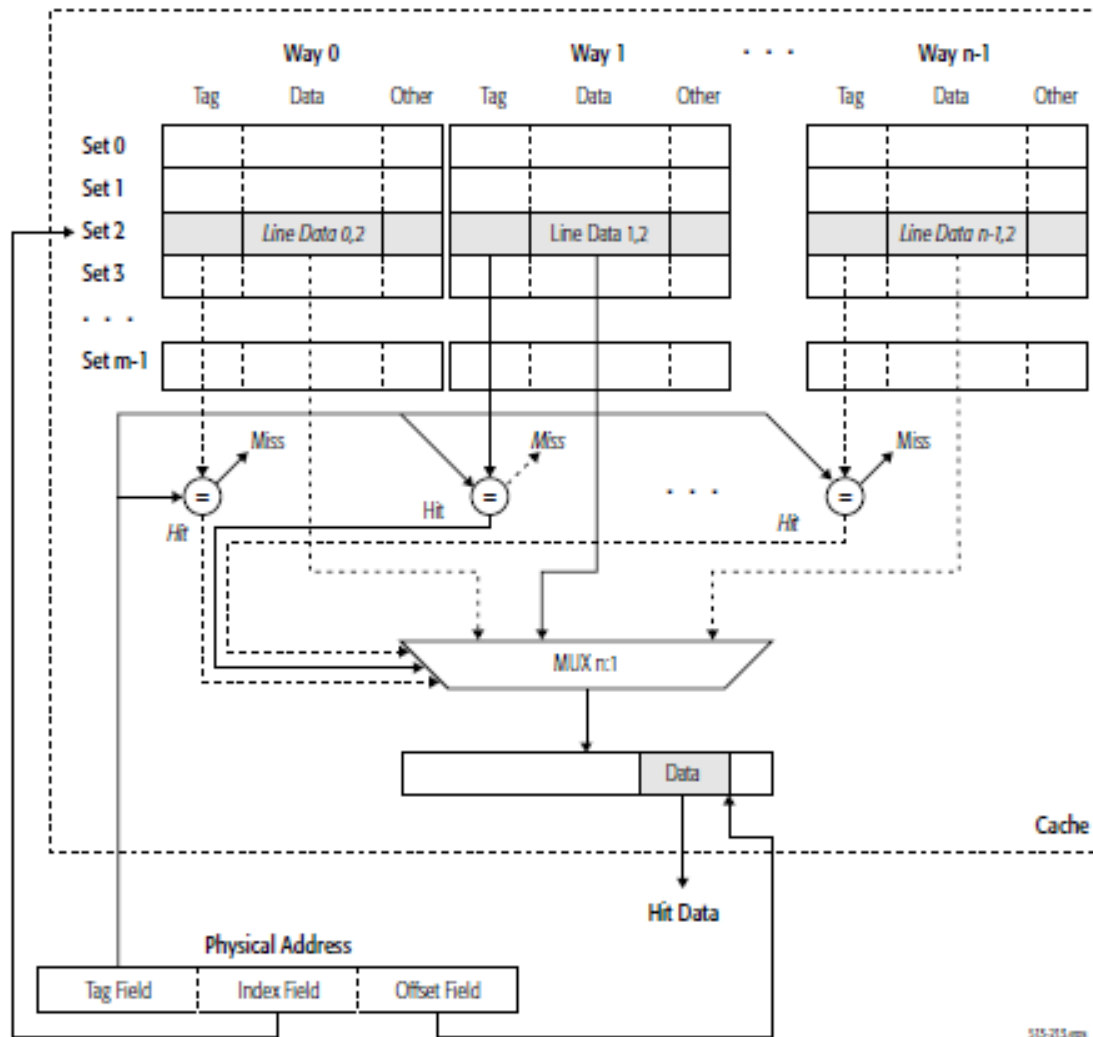


Cache Line Utilization (CLU)

- Idea generated by Lei Yu – IDF filed and in review
- New feature added to AMD CodeAnalyst Software for the Dec. 2011 release
- First step towards providing more data-centric performance analysis
- Intended to measure the relative efficiency of data transfers from main memory to cache
- Relies on Instruction-based Sampling (IBS Op sampling)
- CLU calculated as:
\$line bytes accessed / \$line size



Cache Organization (General Overview)



CLU in AMD CodeAnalyst Software

- AMD CodeAnalyst Software models a separate L1 data cache per core
 - Model gets configuration from `cpuid` (i.e., BD – 16K, 4-way, 64-byte lines)
 - Simple LRU replacement strategy
 - Not a precise model of true L1 data cache – Less than 3% deviation between different configurations
- For each IBS Load and Store op:
 - Break the physical address into Tag, Index + Offset
 - Check for a Tag match in the cache in `set[Index]`
 - Match: Update cache line with bytes accessed
 - No match: Evict the data in the line, tag it with this Tag and update the bytes accessed



CLU in AMD CodeAnalyst Software

- When an eviction occurs (Tag values don't match), update the eviction count for each instruction that previously accessed data in this line
 - Number of evictions increased by 1
 - Total number of bytes in the cache line accessed by this instruction (RIP) is noted
- Lots of data maintained
 - Number of loads, number of stores
 - Total number of loads and stores (in bytes)
 - Physical and Virtual addresses for each load and store
 - Timestamp for each access



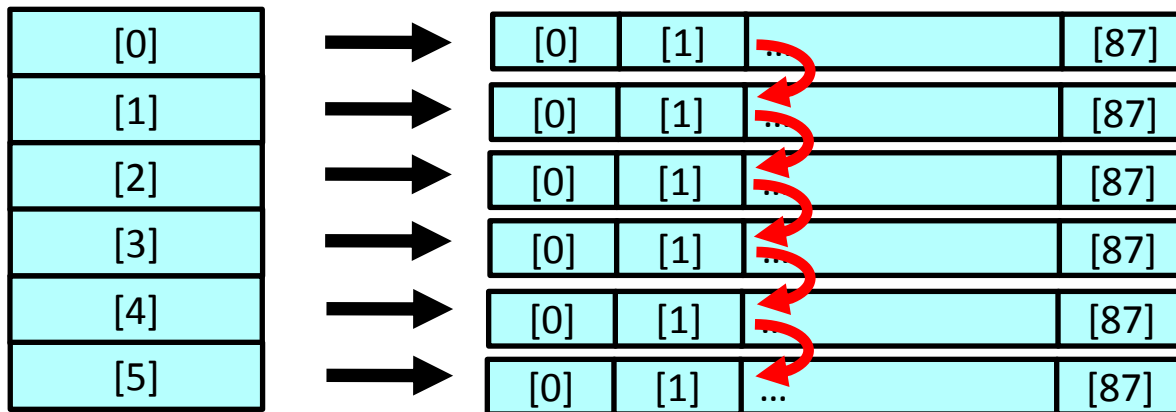
CLU in AMD CodeAnalyst Software

- AMD CodeAnalyst Software collects this data system-wide, for all processes on all cores
- Users can drill down from the process to the module to the function within the module to display source code with associated data
- Each source code line can be expanded to its disassembly for more clarity on identifying issues
- Data is aggregated – i.e., each source line data is the aggregation of its corresponding assembly data



Example: Art benchmark (SPEC[®] CPU2000)

- Art incurs DTLB misses due to long memory strides.
 - Increases chance of eviction for every access

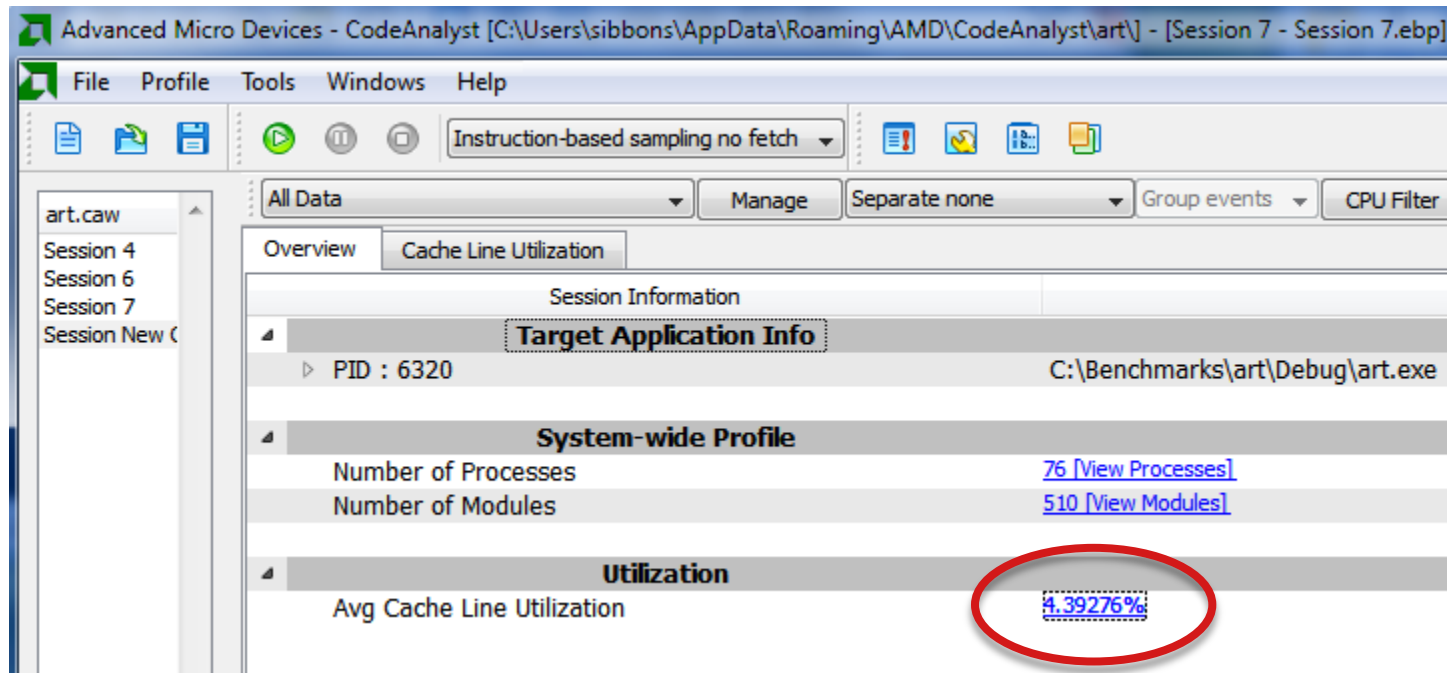


```
for (ti = 0 ; ti < numf1s ; ti++)
    Y[tj].y += f1_layer[ti].P * bus[ti][tj] ;
...
bus = (double **)malloc(numf1s*sizeof(double *));
...
bus[i] = (double *)malloc(numf2s*sizeof(double));
```



Example: Art benchmark (SPEC[®] CPU2000)

- Run a profiling session (IBS Op enabled, CLU enabled)
- Click on the “Avg Cache Line Utilization” hyperlink



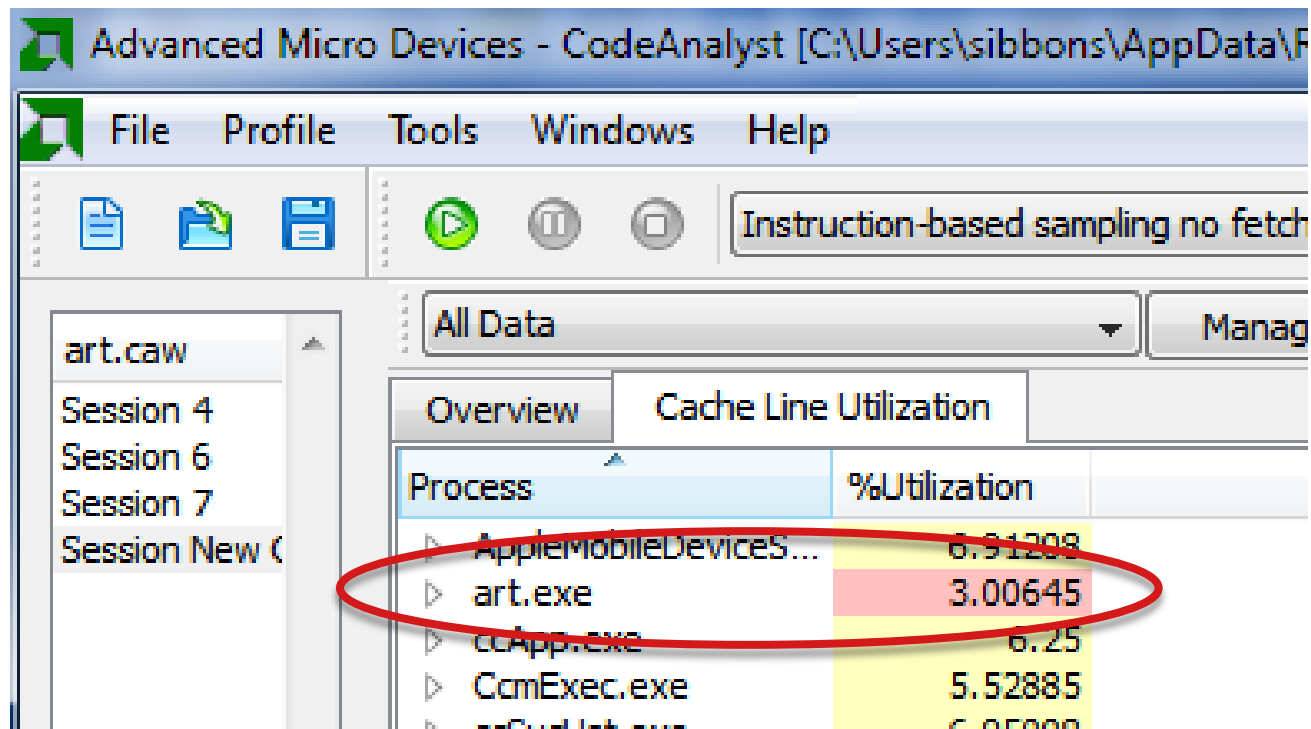
The screenshot shows the AMD CodeAnalyst interface. The main window displays the 'Cache Line Utilization' view for a target application. The 'Utilization' section shows 'Avg Cache Line Utilization' at 4.39276%, which is circled in red. The 'System-wide Profile' section shows 76 processes and 510 modules. The 'Target Application Info' section shows PID 6320 and the path C:\Benchmarks\art\Debug\art.exe.

Section	Item	Value
System-wide Profile	Number of Processes	76 [View Processes]
	Number of Modules	510 [View Modules]
Utilization	Avg Cache Line Utilization	4.39276%
Target Application Info	PID	6320
	Path	C:\Benchmarks\art\Debug\art.exe



Example: Art benchmark (SPEC[®] CPU2000)

- Process tab opens with CLU for all processes
- Expand "art.exe" – will open a sub-list of modules



The screenshot shows the AMD CodeAnalyst interface. The main window is titled "Advanced Micro Devices - CodeAnalyst [C:\Users\sibbons\AppData\F...". The menu bar includes "File", "Profile", "Tools", "Windows", and "Help". Below the menu bar is a toolbar with icons for file operations and a dropdown menu set to "Instruction-based sampling no fetch". The left sidebar shows a tree view with "art.caw" selected, and sub-items for "Session 4", "Session 6", "Session 7", and "Session New (". The main area displays a table with two tabs: "Overview" and "Cache Line Utilization". The "Cache Line Utilization" tab is active, showing a table with the following data:

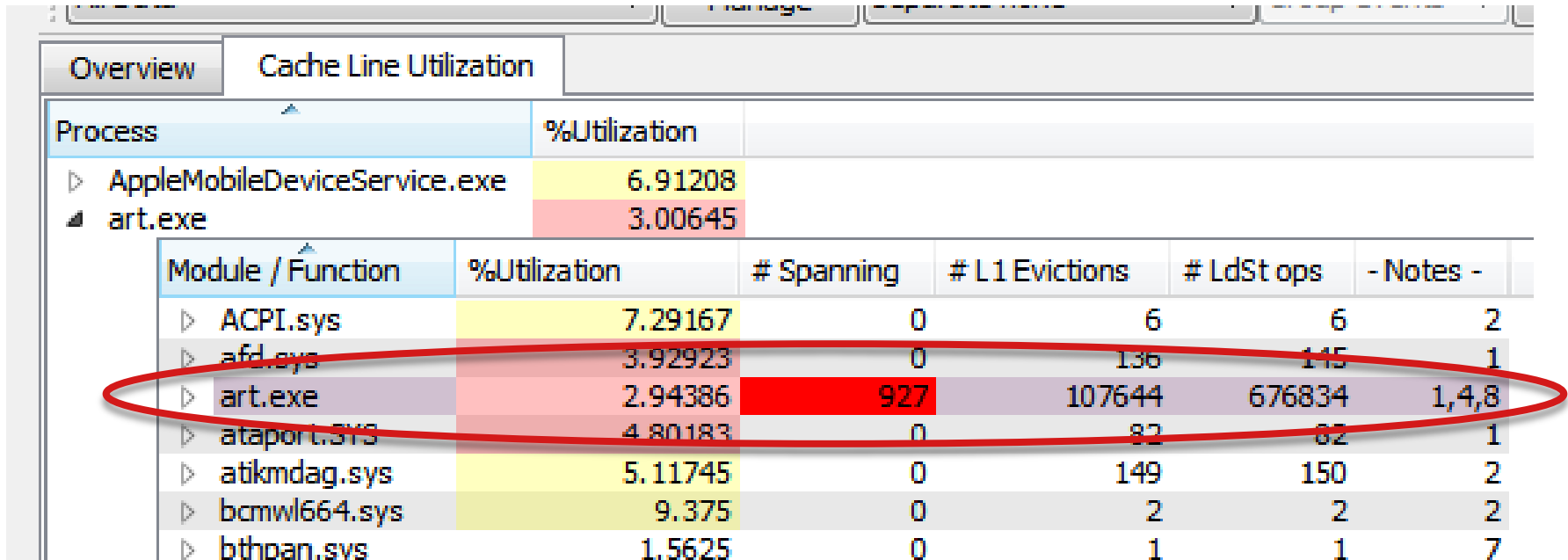
Process	%Utilization
▶ AppleMobileDevicesS...	6.91208
▶ art.exe	3.00645
▶ ccApp.exe	6.25
▶ CcmExec.exe	5.52885
▶ ccSutHat.exe	6.05008

The row for "art.exe" is highlighted in pink and circled in red.



Example: Art benchmark (SPEC[®] CPU2000)

- Expand "art.exe" – will open a sub-list of functions



Process		%Utilization				
▷ AppleMobileDeviceService.exe		6.91208				
▲ art.exe		3.00645				
Module / Function	%Utilization	# Spanning	# L1 Evictions	# LdSt ops	- Notes -	
▷ ACPI.sys	7.29167	0	6	6	2	
▷ afd.sys	3.92923	0	136	145	1	
▷ art.exe	2.94386	927	107644	676834	1,4,8	
▷ ataport.SYS	4.80183	0	82	82	1	
▷ atikmdag.sys	5.11745	0	149	150	2	
▷ bcmwl664.sys	9.375	0	2	2	2	
▷ bthpan.svs	1.5625	0	1	1	7	



Example: Art benchmark (SPEC[®] CPU2000)

- Low cache utilization noted in match and train_match

art.exe 3.00645

Module / Function	%Utilization	# Spanning	# L1 Evictions	# LdSt ops	- Notes -
art.exe	2.94386	927	107644	676834	1,4,8
match	2.41288	0	76725	546645	1,8
train_match	4.86081	0	16084	105997	1
simtest2	3.32452	927	5388	7785	1,4
reset_nodes2	4.9014	0	3258	5090	1
weightadj	3.72468	0	1777	5033	1
simtest	3.57341	0	1352	2194	1
reset_nodes	4.7616	0	991	1189	1
scan_recognize	2.0872	0	944	1184	1
fabs	0.67334	0	362	482	1
_ctrlfp	0.33203	0	320	468	1
g	1.14087	0	252	544	1
_fload_withFB	1.06132	0	53	55	1
sim_other_objects	8.72093	0	43	52	2
sort	2.45711	0	10	10	1

- The simtest2 function has “Spanning” accesses
 - A load or store touches two cache lines



Example: Art benchmark (SPEC[®] CPU2000)

- Opening source view for “match” (double-click on function name) points us to the offending source line

Address	Line	Source	%Utilization	# Spanning	# L1 Evictions	# LdSt ops	- Notes -
	610	/* Compute F2 - y values */					
▷ 0x9728dc	611	for (tj=0;tj<numf2s;tj++)	2.27273	0	11	11	1
	612	{					
▷ 0x9728fd	613	Y[tj].y = 0;					
▷ 0x97290e	614	if (!Y[tj].reset)					
▷ 0x972920	615	for (ti=0;ti<numf1s;ti++)	0.04864	0	1285	216705	1,5,8
▷ 0x97293d	616	Y[tj].y += f1_layer[ti].F * bus[ti][tj];	1.43987	0	21813	119742	1
▷ 0x97297e	617	}					
	618						
	619	/* Find match */					
▷ 0x972983	620	winner = 0;					
▷ 0x97298d	621	for (ti=0;ti<numf2s;ti++)	6.25	0	5	5	2
	622	,					

- This is consistent with the above analysis – The highlighted line is responsible for the long memory strides



Example: Art benchmark (SPEC[®] CPU2000)

- To find where the “Spanning” issues are, open up the source view for simtest2:

▷	0x9713d4	134	su += f1_layer[j].U;	3.2767	0	1030	1466	1
▷	0x9713e9	135	sp += f1_layer[j].P;	1.79907	0	535	1101	1
▲	0x9713ff	136	su2 += f1_layer[j].U * f1_layer[j].U;	3.60317	927	1575	1869	1,4
	0x9713ff		mov eax, [ebp-5ch]	0.21552	0	58	79	1
	0x971402		shl eax, 06h					
	0x971405		mov ecx, [ebp-5ch]	0.13889	0	90	140	1
	0x971408		shl ecx, 06h					
	0x97140b		mov edx, [00475d24h]	0.2193	0	57	87	1
	0x971411		mov esi, [00475d24h]	0.2193	0	57	75	1
	0x971417		fld qword [edx+eax+20h]	11.9932	0	74	74	
	0x97141b		fmul qword [esi+ecx+20h]	9.64506	0	486	487	2
	0x97141f		fadd qword [ebp-38h]	0.06427	472	389	472	1,4
	0x971422		fstp qword [ebp-38h]	0.06868	455	364	455	1,4
▷	0x971425	137	sp2 += f1_layer[j].P * f1_layer[j].P;	3.30711	0	858	1312	1
▷	0x97144b	138	sup += f1_layer[j].U * f1_layer[j].P;	4.86516	0	1029	1454	1
▷	0x971471	139	}					

- This is because the variable “su2” is unaligned



Example: Art benchmark (SPEC[®] CPU2000)

- Analysis notes are displayed to give hints for optimization:

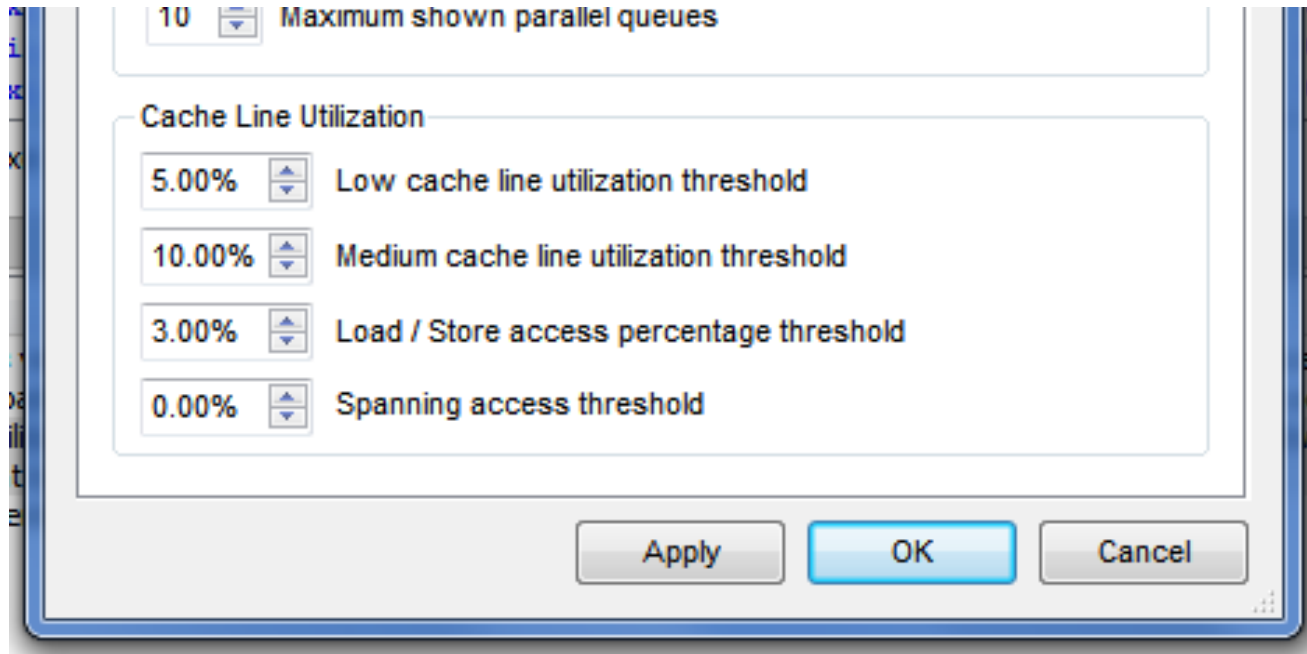
Cache Line Utilization Notes	
Note Number	Description
1	The cache line utilization is very low. This indicates that the data accessed has low spatial locality or the address of the accessed data conflicts with another data access address such that the cache line is evicted shortly after it is loaded. Consider aligning the data or consolidating data such that more data gets loaded into a cache line on initial load.
4	Data accesses span multiple cache lines. This is usually caused by an access to a data element which is not aligned to its natural alignment (i.e., an integer access from an address with the lower 2 bits non-zero). It can also occur when accessing an element of a packed structure.

- Currently, there are 8 separate notes defined
 - 4 for Utilization
 - 2 for Spanning, 1 Compulsory, 1 for bad dasm



Example: Art benchmark (SPEC[®] CPU2000)

- Various thresholds can be set (Options dialog):

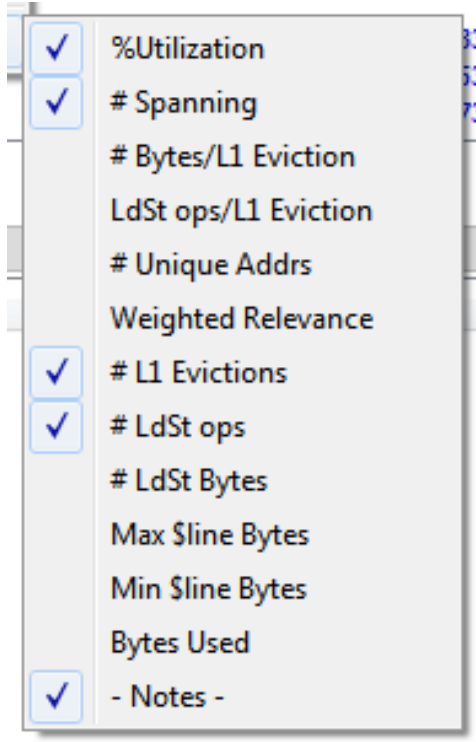


- These control the highlight colors of the cells



Example: Art benchmark (SPEC[®] CPU2000)

■ Data that can be displayed:



- Cache Line Utilization %
- # accesses crossing \$line boundaries
- # bytes accessed per eviction
- # accesses per eviction
- # of data addresses accessed
- "Experimental" weighting factor
- # of evictions
- # of total accesses
- Total # bytes accessed by this instruction
- Maximum # of \$line bytes accessed between evictions
- Minimum # of \$line bytes accessed between evictions
- A 64-bit bitmask indicating which \$line bytes touched
- Descriptive notes for highlighted lines



Future Plans (not committed)

- Perform more user-friendly analysis
 - Narrow down issues to actual variables
 - Suggest alternate structure layouts
 - Etc.
- Use Northbridge data to identify “data false sharing”
- Provide profiling feedback to compilers and linkers for more effective data layout and other data-centric optimizations



Conclusion

- Cache Line Utilization is a first step for AMD CodeAnalyst Software towards more data-centric performance analysis
- CLU provides valuable insight into various aspects of code optimization from a data-centric perspective
- Used in conjunction with other AMD CodeAnalyst Software features, CLU provides powerful analysis of application performance issues



Q & A



The information presented in this document is for information purposes only. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including, but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

▪ **Trademark Attribution**

- AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners. SPEC® is a registered trademark of the Standard Performance Evaluation Corporation.
- ©2011 Advanced Micro Devices, Inc. All rights reserved.

