

Improving Driver Performance – A Worked Example

Bruce Richardson



Legal Disclaimers

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

© 2016 Intel Corporation. Intel, the Intel logo, Intel. Experience What's Inside, and the Intel. Experience What's Inside logo are trademarks of Intel. Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.



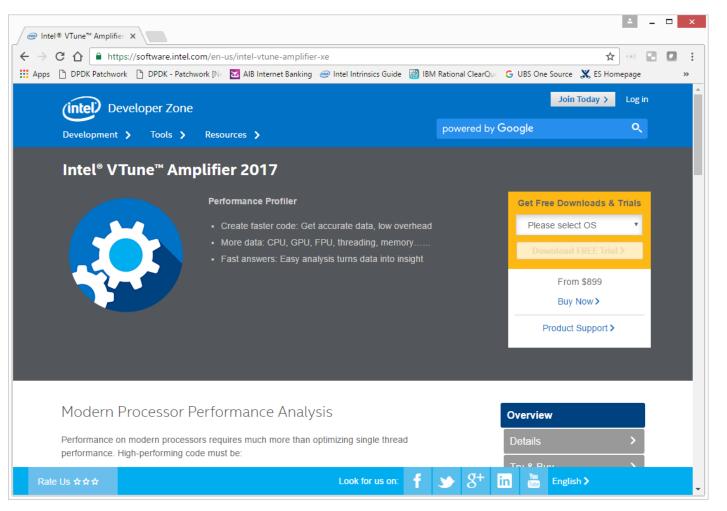
Introduction

- This is a real-world use-case of performance improvement using performance analysis tools – in this case "Intel® VTune™ Amplifier"
- The issues discovered here were largely discovered by accident when investigating other DPDK behaviour
- Hope the walk-through of the issue debug may be of use to others when dealing with performance issues



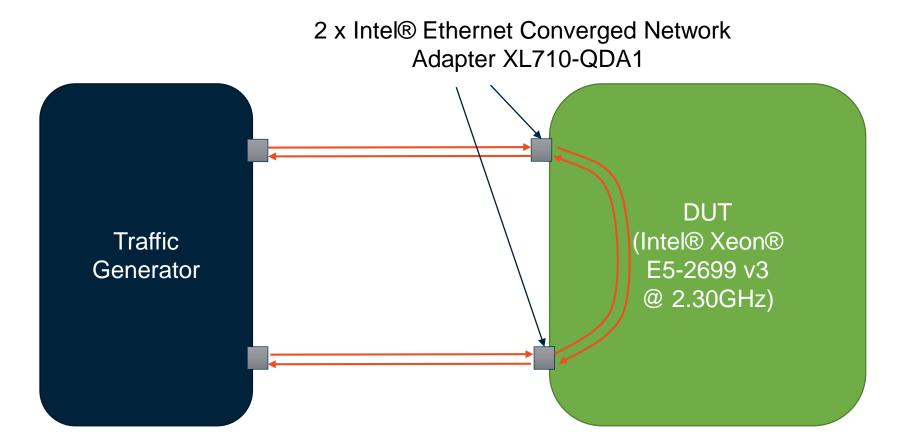
Intel® Vtune Amplifier

- Tool for performance analysis and debugging
- Uses hardware event counters to report on issues affecting program execution, e.g. CPU stalls due to memory access





Test Setup







0.288

This metric shows how often machine was stalled without missing the L1 data cache. The L1 cache typically has the shortest latency. However, in certain cases like loads blocked on older stores, a load might suffer a high latency even though it is being satisfied by the L1. Note that this metric value may be highlighted due to DTLB Overhead or Cycles of 1 Port Utilized issues.

DTLB OV	erhead [@] :	0.000	
Loads Bl	ocked by Store Forwarding [@]	0.189	
identify t	re blocked during store fon he problematically-forwarde han the load, change the st	To streamline memory operations in the pipeline, a load can avoid waiting for memory if a prior store, still in flight, is writing the data that the load wants to read (a 'store	es. Use source/assembly view to identify the blocked loads, then on dynamic instructions prior to the load. If the forwarding store is
<u>Split Loa</u> <u>4K Aliasir</u>		forwarding' process). However, in some cases, generally when the prior store is writing a smaller region than the load is	
<u>FB Full</u> ^② :		reading, the load is blocked for a signficant time pending the store forward. This metric	
This met	ric does a rough estimation	measures the performance penalty of such blocked loads.	ted additional demand L1D demand requests to proceed.
L2 Bound ®		0.000	
L3 Bound	D.	0.179	
	shows how often CPU was s increases performance.	talled on L3 cache, or contended with a sil	oling Core. Avoiding cache misses (L2 misses/L3 hits) improves the
	This metric shows how oft latency and increases perf	en CPU was stalled on L3 cache, or contended with a sibling Core. ormance.	Avoiding cache misses (L2 misses/L3 hits) improves the
	Contested Accesses ⁽⁰⁾ :	0.000	
	Data Sharing [®] :	0.000	
			under unloaded scenarios (possibly L3 latency limited).
	Data Sharing ⁽⁰⁾ : L3 Latency ⁽⁰⁾ :		under unloaded scenarios (possibly L3 latency limited).



(~)

What does this mean?

- We are doing a write to a memory location
- We are subsequently doing a read from that memory location
- The read is getting blocked by the write because the read is for more data than just the write
 - if the write is for the same amount of data, then we can do "store forwarding" to return the data write to the read without hitting cache/mem
 - if the read can be delayed till later, the write can complete and the read can come from cache
- This should not generally show up as a problem in good code



🕼 🛅 🔯 🕨 😺 🕐 😻 Welcome

General Exploration General Exploration viewpoint (change) ③

🕘 Analysis Target 🚔 Analysis Type 📰 Collection Log 🖬 Summary 🤹 Bottom-up 🗳 Event Court 💽 Platform 🖍 HOe_nstr_ve... 🗋 HOe_nstr_ve...

1000ge X

<pre>product product p</pre>	<pre></pre>			*			(2)	1						100	E	-		
<pre>/* 8.1 load 2 mbd point */ descell 2 m_load_sil22('</pre>	<pre></pre>			1000	1265	2555	1. 11.1	1.00			87		1000	1.00	120	-		
<pre>/* 8.1 load 2 mbd point */ descell 2 m_load_sil22('</pre>	<pre></pre>	So. 🔺	Source	Clo.						Address 🛦	Sour.	Assembly					L1.8	louni
<pre>25 /* 8.1 bad 2 shot point '/ 27 desc(1) =(abs, ill(abl, i')(rab + 1); 28 desc(0) =(abs, ill(abl, i')(rab + 1); 29 desc(0) =(abs, ill(abl, i')(rab + 1); 20 desc(0) =(abs, ill(abl, i')(rab, i'); 20 desc(0) =(abs, i')(rab, i'</pre>	<pre> 5</pre>	25			0.000			Ser.			200				opa.		loc L	Split
<pre>227 detecil1 * _m(ask_sil2((_m22: + (mob + 1)); 228 detecil3 * _m(sods_sil2((_m22: + (mob + 1)); 228 detecil3 * _m(sods_sil2((_m22</pre>	<pre>27 decisil =</pre>	326	/* 8 1 load 2 shuf point */					-	0.000	0x52d49f		Black 6:			-	ь	/\$	12007
<pre>222 decs(0) *_m(losd_siz20)(_m220: '*(mcdp)); 223 decs(0) *_m(losd_siz20)(_m220: '*(mcdp)); 224 decs(0) *_m(losd_siz20)(_m220: '*(mcdp)); 225 decs(0) *_m(losd</pre>	<pre>decide =</pre>			1.20	200	6.000	10.0%	0.0%	0.000		341	vpurpckhdo typel, typel, typel						
<pre>229 229 229 229 229 229 229 229 229 229</pre>	<pre>gg / f 1 2 cmy 2 abd past into r_piks */</pre>																	
<pre>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>></pre>	<pre> /* 8 2 cory 2 bud pairs is or nyks */ /* 0 1 pix 1 cory 1</pre>										341							
<pre>mail</pre>	<pre>implify</pre>		/* 8.2 copy 2 abuf point into rg pkts */	-							341						_	
<pre>int split_packet } if split_packet } if split_packet i i split_packet i split_packet i split_packet i split_packet i split_packet i i split_packet i split_pac</pre>	<pre>product if (mplit_mextet) { for (mplit_mextet) { for (mplit_mextet) { for (mplit_mextet) {</pre>														-			
<pre>## fight_macket) { ret_prefetch0[sr_pkts[post-1-cachelinal]; ret_prefetch0[sr_pkts[post-1-cachelinal]; ret_prefetch0[sr_pkts[post-1]-cachelinal]; ret_prefetch0[sr_pkts[post-1]-cachelinal];</pre>	<pre>33 af (mplingsacket) {</pre>																	
<pre>def = respretetol(arg_stations); respreteto</pre>	<pre>bit = respected/org_kisles=:2:=exhilasi; respected/org_kisles=:exhilasi; respected/org_kisles::=exhilasi; respected/org_kisles::=e</pre>		if (solit packet) {	1.20	1.00	1,200	0.0%	D.0%	0.000									
<pre>prime for the format from desc to pktmbuf *</pre>	<pre>05</pre>																	
<pre>best rts_prefetch016rs_ptts1pes + 21-scabeline11; rts_prefetch016rs_ptts1pes + 21-scabeline11; rts_complet_bartist2; rts_complet_bartist2; rts_complet_bartist2; rts_complet_bartist2; rts_complet_bartist2; rts_complet_bartist2; rts_complet_bartist2; rts_complet_bartist2</pre>	<pre>bit re_predetoid/r_gitige : 2]-scathing1; re_predetoid/r_gitige : 2]-scathing2; re_predetoid/r_gitige : 2]-scathing2; re_predet</pre>																	
<pre>int_prefetcholdsr_pittalpos + 31-xcachelinal); /*shift the pitle field*/ dec_pitler_align(deccs); /*shift the pitle field*/ dec_pitler_align(deccs); /* avoid compiler_reorder optimization */ rte_compiler_barrier1; /* avoid compiler_reorder optimization */ rte_compiler_barrier1; /* avoid compiler_string */ /*shift the pitle stater info only */ pit_mb = _mm_shuffle_epi8(deccs[3], deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _m_opackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */ star_tupe 2 = _mopackli_epi8(deccs[3], shuf_msk); /* C1 4 +>2 filter stater info only */</pre>	<pre>int =</pre>														-			
<pre>>> /* /*#vift the pitlen field*/ desc_pitlen_alsyndescs; /* state parties_intervient; /* void compiler_reorder optimization */ rte_compiler_barrier(); /* D.1 pkt 3,4 convert format from desc to pktmbuf * _mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); /* Lab 2,4 convert format from desc to pktmbuf * _mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); /* vmovdqax -0x18(%rsp), %xmm1 /* vmovdqax -0x28(%rsp), %xmm1 /* vmovdqax -0x28(%rsp),</pre>	<pre> **</pre>				-					a provide and and and the set of the					-		_	
<pre>39 90 91 92 92 94 94 94 94 94 94 94 94 94 94 94 94 94</pre>	<pre> /*#viit the jailer field// des_sition_align(desci); /* woid compiler rearder spiniszation */ reaction_align(desci); /* usid compiler rearder spiniszation */ reaction_align(desci); /* usid compiler rearder spiniszation */ reaction_align(desci); /* 10 # 10 # 10 # 10 # 10 # 10 # 10 # 10</pre>		}							and the second sec								
<pre>40 /* bift the pitter file/*/ 41 desc.pitten_ilightdescs); 42 /* woid compiler reorder optimization */ 42 /* woid compiler reorder optimization */ 44 /* woid compiler reorder optimization */ 45 /* 0.1 plt 3, 4 convert format from desc to pltmbuf */ 45 /* 0.1 plt 3, 4 convert format from desc to pltmbuf * 46 /* 0.1 plt 3, 4 convert format from desc to pltmbuf * 47 /* 0.1 plt 3, 4 convert format from desc to pltmbuf * 48 /* woid compiler reorder woid (vrsp), tamed 49 /* woid compiler reorder woid (vrsp), tamed 49 /* 0.1 plt 3, 4 convert format from desc to pltmbuf * 49 /* 0.1 plt 3, 4 convert format from desc to pltmbuf * 40 /* 0.1 plt 3, 4 convert format from desc to pltmbuf * 41 /* woid compiler reorder woid (vrsp), tamed 41 /* woid compiler reorder woid (vrsp), tamed 42 /* woid compiler reorder woid (vrsp), tamed 42 /* woid compiler reorder woid (vrsp), tamed 42 /* woid compiler reorder woid (vrsp), tamed 43 /* woid compiler reorder woid (vrsp), tamed 44 /* woid compiler reorder woid (vrsp), tamed 45 /* woid compiler reorder woid (vrsp), tamed 45 /* woid compiler reorder woid (vrsp), tamed 45 /* woid (vrsp), tamed 46 /* woid (vrsp), tamed 46 /* woid (vrsp), tamed 47 /* wo</pre>	<pre>d0 /*shift bagklen faid/ d1 dec_pathwal_tapid(decs); /* avoid compiler reorder optimization */ rfs_compiler_hard(decs); /* avoid compiler reorder optimization */ rfs_compiler_hard(decs); /* 0.1 pkt 3,4 convert format from desc to pktmbuf * mb3 = _mm_shuffle_epi8(descs[3], shuf_msk); /* 1 hos iffs_sidescs[1], shuf_msk); mb3 = _mm_shuffle_epi8(descs[2], shuf_msk); /* 21 hos iffs_sidescs[1], shuf_msk]; /* 2</pre>																	
<pre>44 desc_pktlen_align(descs); 42 /* woid compiler reorder optimization */ 44 /* to_i pkt side _ emstuffle_epi8(descs[3], shuf_msk); 45 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 46 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 47 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 48 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 49 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 40 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 40 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 40 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 40 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 40 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 41 /* 0 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 41 /* 0 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 41 /* 0 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 41 /* 0 /* 0.1 pkt 3,4 convert format from desc to pktmbuf */ 41 /* 0 /* 0 /* 0 /* 0 /* 0 /* 0 /* 0 /*</pre>	<pre>desc_stilestign(descs); /* word compiler reorder optimization */ ret_compilestring(descs); /* to i pit h, convert format from desc to pktmbd */ pt_dt_dt =</pre>		/*chift the oktion field*/					-					-					
<pre>42 42 42 43 44 44 44 44 44 44 44 44 44 44 44 44</pre>	<pre>44 44 45 45 46 46 46 47 47 48 46 47 48 46 47 48 46 47 48 47 48 48 48 48 48 48 48 48 48 48 48 48 48</pre>																	
<pre>/* avoid compiler reorder optimization */ ret_compiler_harrier[]</pre>	<pre>/* suid compler render optimization */ ** suid compler render optimization */ ** suid compler device optimization */ ** suid complet device optimization ** substant ** substan</pre>		one-buccardarabulones);							and the state of the second se							_	
<pre>##</pre>	<pre>starter_tex_compler_barcater(); /* D1 pkt 3,4 convert format from desc to pktmbd */ pkt_bd =mw_useht(=_epi8(descs[2], shuf_msk); pbt_bd =mm_shuffle_epi8(descs[2], shuf_msk); mb3 = _mm_shuffle_epi8(descs[2], shuf_msk); /* C2 epi 4 pkt stater value */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); mb3 = _mm_shuffle_epi8(descs[2], shuf_msk); /* C2 epi 4 pkt stater value */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); mb3 = _mm_shuffle_epi8(descs[2], shuf_msk); /* C2 epi 4 pkt stater value */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C2 epi 4 pkt stater value */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ pt_bd =mw_useht(=_epi8(descs[2], shuf_msk); /* C3 copy final 3,4 data to rupkts */ /* C3</pre>		(# world compiler containsting #/								and the second second second		1000	0.0%	1 25 24	0.000		
<pre>bit /* D.1 pkt 3,4 convert format from desc to pktmbuf */ pkt_mb1 = mim_shuffle_epi8(descs[3], shuf_msk);</pre>	<pre>65 // 0.1 pkt 3,4 convert format from desc to pktmbuf */ pt_wish = _m_sduffix_esildesc(3), shuf_wk); 66 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[3], shuf_msk); 67 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[3], shuf_msk); 67 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 68 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 69 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 60 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 60 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 60 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 61 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 62 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 62 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 63 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 64 /* C1 4+2 filter stater info only */ ster_may = _mshuffile_epi8(descs[2], shuf_msk); 65 /* C2 yet into stater info only */ 66 /* C2 yet into stater info only */ 66 /* C2 yet into stater info only */ 66 /* C2 yet into stater info only */ 66 /* C2 yet into stater info only */ 66 /* C2 yet into stater info only */ 66 /* C2 yet into stater info only */ 66 /* C2 yet info stater info only */ 67 /* C2 yet info stater info only */ 67 /* C2 yet info stater info only */ 68 /* C2 yet info stater info only */ 68 /* C2 yet info stater info only */ 68 /* C2 yet info stater info only */ 69 /* C2 yet info stater info only */ 69 /* C2 yet info stater info only */ 69 /* C2 yet info stater info only */ 69 /* C2 yet info stater info only */ 69 /* C2 yet info stater info only */ 69 /* C2 yet info stater info only */ 69 /* C2 yet info stater info only */ 69 /* C2 y</pre>				-					the second state of the se			and the second second					
<pre>/* Cl #>2 filter stater info only */ sterr_tmp2 * _mm_unpackhi_epi32(descs[3], descs[2]); 0.1 pkt 3,4 convert format from desc to pktmbuf * _mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); </pre>	<pre>/*Clevefilterstater info only */ ster.tmp2 * ms.ympchbi.sml2(dece[1], dece[2]); **Clevefilterstater info only */ ster.tmp2 * ms.ympchbi.sml2(dece[1], tsml2(dece[1]); **Clevefilterstater info only */ mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); mb3 = _mm_shuffle_epi8(descs[2], shuf_msk); **Clevefilterstater info only */ mstresselled(cold */ **Clevefilterstater info only */ mstresselled(cold */ **Clevefilterstater value */ ***********************************</pre>	47	<pre>pkt_mb4 = _mm_shuffle_epi8(descs[3), shuf_msk);</pre>	520	1,00					0×52d4fd	70	and structure in the state of t	100	0.0%	0.0%	0.000	1.000	
<pre>bit sterr_tup2 = _ec_uppekhi_epi2(descs[3], descs[2]); D.1 pkt 3,4 convert format from desc to pktmbuf * _mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); </pre>	<pre>bit sterr twp2 * _mu.uppackbi.gei32(desc(3), descs(2)); c) 1 pkt 3,4 convert format from desc to pktmbuf * _mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); _mb3 = _mm_shuffle_epi8(descs[2], shuf_msk); c) 2 pt_mb2 * _mu.genuffle_epi8(descs[2], shuf_msk); c) 2</pre>	0				2.200	-	0.014		= 0x52d507	343	vpshufb %xmm3, %xmm1, %xmm5						
D.1 pkt 3,4 convert format from desc to pktmbuf * vmovdqax -0x18(%rsp), %xmm11 _mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); vmovdqax -0x28(%rsp), %xmm1	<pre>D.1 pkt 3,4 convert format from desc to pktmbuf * _mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); _mb3 = _mm_shuffle_epi8(descs[2], shuf_msk); _vmovdqax -0x28(%rsp), %xmm1 vmovdqax -0x48(%rsp), %xmm1 vmovdqax -0x48(%rsp), %xmm1 vmovdqax -0x48(%rsp), %xmm3 vmovdqax -0x48(%rsp), %xmm3</pre>	50								0-52-50-		vpuppekhdo Symal Symall Symal						
D.1 pkt 3,4 convert format from desc to pktmbuf * _mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); vmovdqax -0x18(%rsp), %xmm1	D.1 pkt 3,4 convert format from desc to pktmbuf * _mb4 = _mm_shuffle_epi8(descs[3], shuf_msk); _mb3 = _mm_shuffle_epi8(descs[2], shuf_msk);										25							
	bit pit_mb2 * _mm_phuffle_mp18(descs[1], shuf_msk); pit_mb1, shuf_msk); pit_mb2 * _mm_phuffle_mp18(descs[0], shuffle_mp18(descs[0],										20							
	05 /* C.2 get 4 pkts staterr value */ 0x52d54e 363 vpshufb %xmm3, %xmm3 066 zero = _mm_xor_sil28(d_check, dd_check); 0x52d54e 363 vpshufb %xmm3, %xmm3 067 staterr = _mm_urpacklo_epi82(sterr_tmp1, sterr_tmp2); 0x52d55 376 vpaddw %xmm3, %xmm3 068 0x52d55 377 vpaddw %xmm3, %xmm3, %xmm3 0xmm4 069 /* 0.3 copy final 3,4 data to rx_pkts */ 0x52d55a 377 vpaddw %xmm3, %xmm3, %xmm3 070 _mm_storeu_sil28((void *)6rx_pkts[pos+3].>rx_descriptor_fields1, 0x52d55a 855 movq %rbp, 0x88(%r10) 071 _pkt_mb4); 0x52d553 355 movq %rbp, %r0	D.1mb4 mb3	<pre>pkt 3,4 convert format from des = _mm_shuffle_epi8(descs[3], s = _mm_shuffle_epi8(descs[2], s </pre>	shu	£_m	_ .sk));	uf	*	0-52d5 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d	mov mov	<pre>vpurpthvd tunel, tunel, tunel vdqax -0x18(%rsp), %xmm11 vdqax -0x28(%rsp), %xmm1 vdqax -0x48(%rsp), %xmm3</pre>						
	b66 zero = _mm_xor_sil28(d_check, dd_check); 0x52d553 376 vpaddw %xmm4, %xmm7, %xmm4 b67 staterr = _mm_wopacklo_epi82(sterr_tmp1, sterr_tmp2); 0x52d557 355 movzx %dx, %ubp b68 0x52d55a 977 vpaddw %xmm3, %xmm7, %xmm3 0x52d55a 977 b69 /* 0.3 copy final 3,4 data to rx_pkts */ 0x52d55a 977 vpaddw %xmm3, %xmm3 0x52d55a b70 _mm_storew_sil28((void *) 6rx_pkts[pos+3] >rx_descriptor_fields1, 0x52d55a 855 movq %vbp, 0x18(%r20) b71 _pkt_mb4); 0x52d553 355 movq %vbp, %r20 0x52d55a	D.1 <u></u> mb4 _mb3	<pre>pkt 3,4 convert format from des = _mm_shuffle_epi8(descs[3], s = _mm_shuffle_epi8(descs[2], s </pre>	shu	£_m	_ .sk));	uf	*	0-52d5 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d5 0-52d545	mov mov	<pre>vpurpthvd turel, turel, turel vdqax -0x18(%rsp), %xmm11 vdqax -0x28(%rsp), %xmm1 vdqax -0x48(%rsp), %xmm3 vdqax -0x48(%rsp), %xmm3</pre>						
0x52x549 355 vmovq %xma2, %rdx	067 staterr = _m_urpscklo_epi32(sterr_tsp), sterr_tsp2); 0x52d557 355 movzx %dx, %ebp 068 0x52d55a 077 vppddw %xmm3, %xmm3 0x52d55a 077 069 0x52d55a 077 vppddw %xmm3, %xmm3 0x52d55a <	D.1 _mb4 _mb3	<pre>pkt 3,4 convert format from des = _mm_shuffle_epi8(descs[3], s = _mm_shuffle_epi8(descs[2], s pkt_moz = _mmffle_epi8(descs[1], shuf_msk); </pre>	shu	£_m	_ .sk));	uf	*	0-52855 0-528 0-528 0-528 0-528 0-528 0-528 0-528 0-528 0-528 0-528 0-5285 0-528540	mov mov	<pre>vpurpthvd tumel, tumel, tumel vdqax -0x18(%rsp), %xmm11 vdqax -0x28(%rsp), %xmm1 vdqax -0x48(%rsp), %xmm3</pre>						
0x52x3549 355 vmovq %xmm2, %rdx 365 /* C.2 get 4 pkts staterr value */ 0x52x354e 363 vpshufb %xmm3, %xmm3	068 0x52d55a 877 vpaddw %xmm3, %xmm7, %xmm3 069 /* 0.3 copy final 3,4 data to rx_pkts */ 0x52d55a 857 vpaddw %xmm3, %xmm7, %xmm3 070 _mm_storew_sil28((void *)@rx_pkts(pos+3)->rx_descriptor_fields1, 0x52d55a 855 movq (%rs,0) %xmm7, %xmm3 071 _pkt_mb4); 0x52d553 355 movq (%rs,0) %xm7, %xmm3	D.1 _mb4 _mb3	<pre>pkt 3,4 convert format from des = _mm_shuffle_epi8(descs[3], s = _mm_shuffle_epi8(descs[2], s pkt_mod = _mon_shuffle_epi8(descs[1], shuf_mod) pkt_mod = _mon_shuffle_epi8(descs[1], shuffle_epi8(descs[1], shuffle_e</pre>	shu	£_m	_ .sk));	uf	*	0+52d5 0+52d 0+52d 0+52d 0+52d 0+52d 0+52d 0+52d 0+52d 0+52d 0+52d545 0+52d545 0+52d545	mot mot 355 355 363	<pre>vpurptkhvd tumel, tumel, tumel vdqax -0x18(%rsp), %xmm11 vdqax -0x28(%rsp), %xmm1 vdqax -0x48(%rsp), %xmm3 vportume2, tume2, tume2 vportume2, tume2, tume2 vportume2, tume2, tume2</pre>						
054 0x52d549 355 vmovq %xmm2, %rdx 165 /* C.2 get 4 pkts staterr value */ 0x52d54e 363 vpshufb %xmm3, %xmm3, %xmm3 166 zero = _mm_xor_sil28(dd_check, dd_check); 0x52d553 376 vpaddw %xmm4, %xmm7, %xmm4	/* 0.3 copy final 3,4 data to rx_pkts */ 0x52d55e 855 movq %rbp, 0x18(%r10) /70 _mm_storeu_si128((void *)℞_pkts[pos+3]->rx_descriptor_fields1, 0x52d562 355 movq %rbp, 0x18(%r10) /71 _pkt_mb4); 0x52d565 355 movq %rbx, %r10	D.1 _mb4 _mb3	<pre>pkt 3,4 convert format from des = _mm_shuffle_epi8(descs[3], s = _mm_shuffle_epi8(descs[2], s pkt_sol = _m_shuffle_epi8(descs[2], s pkt_sol = _m_shuffle_epi8(descs[0], shuf_esk); /* C.2 get 4 pkts staterr value */ zero = _m_xor_sil28(dd_check, dd_check);</pre>	shu	£_m	_ .sk));	uf	*	0-52d5 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d540 0-52d545 0-52d545 0-52d545	mot mot 355 355 363 376	<pre>vpurpckhvd tumel, tumel, tumel vdqax -0x18(%rsp), %xmm11 vdqax -0x28(%rsp), %xmm1 vdqax -0x48(%rsp), %xmm3 vportsem2, tumel, tume2 vpor tume2, tumel, tume2 vportume2, tumel, tume3 vpodut tumed, tume3, tume3</pre>						
064 0x52d549 355 vmovq %xmm2, %rdx 055 /* C.2 get 4 pkts staterr value */ 0x52d54 963 vpshufb %xmm3, %xmm3 066 zero = _mm_xor_sil28(dd_check); 0x52d553 376 vpsdudv %xmm7, %xmm4 067 staterr = _mm_urpscklog_epi82(sterr_tmp2); 0x52d557 355 novzx %dx, %upp	I/O _mm_storeu_sil28((void *)℞_pkts[pos+8]->rx_descriptor_fields1, 0x52d562 355 movq (%rsi), %rbp /71 pkt_mb4); 0x52d563 355 mov %rdx, %r10	D.1 _mb4 _mb3	<pre>pkt 3,4 convert format from des = _mm_shuffle_epi8(descs[3], s = _mm_shuffle_epi8(descs[2], s pkt_sol = _m_shuffle_epi8(descs[2], s pkt_sol = _m_shuffle_epi8(descs[0], shuf_esk); /* C.2 get 4 pkts staterr value */ zero = _m_xor_sil28(dd_check, dd_check);</pre>	shu	£_m	_ .sk));	uf	*	0-528557 0-528 0-558 0-558 0-558 0-558 0-558 0-558 0-558 0-558 0-558 0-558 0-5	mov mov 355 355 363 376 355	<pre>vpurptkhvd tumel, tumel, tumel vdqax -0x18(%rsp), %xmm11 vdqax -0x28(%rsp), %xmm1 vdqax -0x48(%rsp), %xmm3 vportume2, tume2, tume2 vmovq tume2, tume1, tume2 vmovq tume2, tume1, tume3 vpodth tume4, tume3, tume3</pre>						
64 0x52d549 355 vmovq %xmm2, %rdx 65 /* C.2 get 4 pkts staterr value */ 0x52d54 363 vpahufb %xmm3, %xmm3 66 zero = _mm_xor_sil28(dd_check, dd_check); 0x52d55 376 vpaddv %xmm3, %xmm3 67 staterr = _mm_unpacklo_epi32(sterr_tep), sterr_tmp2); 0x52d55 377 vpaddv %xmm3, %xmm3 68 0x52d55a 377 vpaddv %xmm3, %xmm3 xmm7, %xmm3	71 pkt_mb4); 0x52d565 355 mov %rdx, %r10	D.1 (_mb4 _mb3	<pre>pkt 3,4 convert format from des = _mm_shuffle_epi8(descs[3], s = _mm_shuffle_epi8(descs[2], s pkt_mol = _mshuffle_epi8(descs[1], shuf_msk); pkt_mol = _mshuffle_epi8(descs[0], shuf_msk); /* C.2 get 4 pkts staterr value */ zero = _ms_xor_sil28(dd_check, dd_check); staterr = _ms_urpacklo_epi82(sterr_tepl, sterr_tep2);</pre>	shu	£_m	_ .sk));	uf	*	0-52d5 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d5 0-52d545 0-52d545 0-52d545 0-52d545 0-52d558	mor mor 355 355 363 376 355 377	<pre>vpurpckhvd tumel, tumel, tumel vdqax -0x18(%rsp), %xmm11 vdqax -0x28(%rsp), %xmm1 vdqax -0x48(%rsp), %xmm3 vpotuto tumed, tumel vmovg tumed, tumel, tumed vpotuto tumed, tumel, tumed vpotuto tumed, tumed, tumed</pre>						
64 0x52d549 355 vmovq %xmm2, %rdx 65 /* C.2 get 4 pkts staterr value */ 0x52d54 963 vpshufb %xmm3, %xmm3, %xmm3 66 zero = _mm_xor_sil28(dd_check, dd_check); 0x52d55 376 vpsddv %xmm4, %xmm7, %xmm4 67 staterr = _mm_urpacklo_epi32(sterr_tmp1, sterr_tmp2); 0x52d55 375 movzx %dx, %ebp 68 0x52d55a 377 vpaddv %xmm3, %xmm3, %xmm3 0x52d55a 69 /* 0.3 copy final 3,4 data to rx_pkts */ 0x52d55a 855 movq %rbp, 0x18(%r20)		51 D.1 _mb4 _mb3	<pre>pkt 3,4 convert format from des = _mm_shuffle_epi8(descs[3], s = _mm_shuffle_epi8(descs[2], s pkt_mbl = _m_shuffle_epi8(descs[2], s pkt_mbl = _m_shuffle_epi8(descs[0], shuf_msk); /* C.2 get 4 pkts staterr value */ zero = _me_xor_sil28(dd_check, dd_check); staterr = _m_urgacklo_epi82(sterr_tmpl, sterr_tmp2); /* 0.3 copy final 3,4 data to rapkts */</pre>	shu	£_m	_ .sk));	uf	*	0+52853 0+52855 0+52855 0+52855 0+52855 0+52855 0+52855 0+528555 0+558555 0+5285555 0+528555 0+5585555 0+5285555 0+5285555 0+5285555 0+5285555 0+5285555 0+52855550 0+52855550000000000000000000000000000000	mov mov 355 355 363 376 355 377 355	<pre>vpurpckhvd tumel, tumel, tumel vdqax -0x18(%rsp), %xmm11 vdqax -0x28(%rsp), %xmm1 vdqax -0x48(%rsp), %xmm3 vpofuto tume2, tume2 vpor tume2, tume2, tume2 vpor tume2, tume3, tume3 vpaddw tume4, tume7, tume4 movg tupe, 0x18(tr20)</pre>						
64 0x52d54 355 vmovq %xmm2, %rdx 65 /* C.2 get 4 pkts staterr value */ 0x52d54 863 vpohu/b %xmm3, %xmm3, %xmm3 66 zero = _mm_xor_sil28(d_check, dd_check); 0x52d54 863 vpohu/b %xmm3, %xmm3, %xmm3 67 staterr = _mm_urpacklo_epi32(sterr_tmp1, sterr_tmp2); 0x52d55 376 vpaddv %xmm3, %xmm3 68		51 _mb4 _mb3	<pre>pkt 3,4 convert format from des = _mm_shuffle_epi8(descs[3], s = _mm_shuffle_epi8(descs[2], s pkt_sol = _m_shuffle_epi8(descs[2], s pkt_sol = _m_shuffle_epi8(descs[0], shuf_esk); /* C 2 get 4 pkts staterr value */ zero = _m_sor_sil28(d_check, d_check); staterr = _m_urpacklo_epi82(sterr_tep), sterr_tep2); /* 0.3 copy final 3,4 data to rs_pkts */ _m_storeu_sil28((void *)&rs_pkts[pos+3].>rs_descriptor_fields],</pre>	shu	£_m	_ .sk));	uf	*	0-52d52 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d 0-52d540 0-52d540 0-52d540 0-52d545 0-52d553 0-52d554 0-52d555 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d5 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d55 0-52d5 0-52d55 0-52d55 0-52d55 0-52d55 0-52d5 0-52d55 0-52d5	mov mov 355 355 363 376 355 376 355 377 855 375	<pre>vpurpckhvd tumel, tumel, tumel vdqax -0x18(%rsp), %xmm11 vdqax -0x28(%rsp), %xmm1 vdqax -0x48(%rsp), %xmm3 vpotufb tumel, tumel vpor tumel, tumel, tumel vpotufb tumel vpotu</pre>						



Intel VTune Amplifier XE 2016

Reading this case

- In this case, both the instruction highlighted and the previous one are 128-bit loads.
- Therefore either one is a potential candidate for the source of the delay
- If we assume that this is the read, then we need to find the offending write:
 - occurs previous to these [nice and easy to find in C, as there is a compiler barrier]
 - is of a size smaller than 128-bits



Other Weirdness...

• Why does the code:

pkt_mb3 = _mm_shuffle_epi8(descs[2], shuf_msk); cause a read from memory at all?

- Shouldn't descs[2] have already been loaded to xmm register previously at the line?
 descs[2] = _mm_loadu_sil28((__ml28i *)(rxdp + 2));
- Let's look at that previous load lines in vtune...







What is happening?

- A load instrinsic is resulting in an xmm load followed by a store?
- We have a second mystery.
- Let's trace back through what happens to the descriptors through the code between the two points...



desc_pktlen_align

- Only work done between desc[2] load and the offending line is function "desc_pktlen_align"
- Again look at assembler listing



	inalysis Target 🖳 Analysis Type 🔄 Collection Log 🚹 Summary 🚱 Bottom-up 🚱 Even Assembly 🛐 🥅 🤯 🚳 🦥 🦓 🌒 Q. Assembly prouping:	Contraction in the		atform	b 140	e_ritk_	ve	3 140e_rxtx31			All and	- 11
Source	Assembly 📰 🛅 🦥 🍩 🕸 😧 🔍 Assembly grouping	Addre		1	1	DE	1		1	T.	0 0	
•	Source	Clo	Instr Retir	CPI Rate	Fro. Bo	Ba. Sp.	DTLB	Address	Sour Line	Assembly	Pi Fro. Bad Ate Bo. Spe. DTL Over	
	* D. fill info. from desc to abut			-			10000	0×526489	335	movg (%rsi), %rdx		by S
	*/							0×52d48c	335	prefetcht0z 0x40(%rdx) Vector Code		
		Sugar.		a second				0×52d450	336	movq OxB(%rsi), %rdx		
_	for (pos = 0, nb_pkts_recd = 0; pos < RTE_I40E_VPMD_RX_BURGT;	2,80	200	14.000	2.0%	2.24	0.00		336	prefetcht0z 0x40(%rdx)		
-	pos += RTE_I40E_DESCS_PER_L00P,	7.60			0.04			0×526498	337 337	movg (%r8), Ardx		
	<pre>rxdp += RTE_I40E_DESCS_PER_L00P) {w128i_descs[RTE_I40E_DESCS_PER_L00P];</pre>	7.60			15.15.46		0,000	0x52d49b 0x52d49f	337	prefetcht0z 0x40(%rdx)		
-	mizes_descs(rke_ivve_descs_rer_cvvr), mizes_pkt_mbl, pkt_mb2, pkt_mb3, pkt_mb4;							0x52d49f	341	vpurpckhdg %xmml, %xmm0, %xmml		- i i
	ml28i zero, staterr, sterr_tmpl, sterr_tmp2;		-					0x52d4a3	341	vpurpckhdg Nxmill, Nxmill, Nxmill		
	_m128i mbp1, mbp2; /* two mbuf pointer in one XMM reg. */		-					0x52d4a7	341	vpunpckhdg %xmm0, %xmm1, %xmm0		
								0x52d4ab	341	vparld \$0x6, %xmm0, %xmm0		
	/* B.1 load 1 mbuf point */							0x52d4b0	341	vpand %xxxx0, %xxxx15, %xxxx0		
	<pre>shpl = _em_loadu_sil28((ml28i *)&sw_ring[pos]);</pre>							= 0x52d4b4	341	vpackssdv Sveell, Sveell, Sveell		
-	/* Read desc statuses backwards to avoid race condition */		-					0x52d4b9	341			4
-	/* Alload 4 pkts desc */	2.00	1.00					0x52d4be	341			
	<pre>descs[3] = _mm_loadu_sil20((ml20i *)(rxdp + 3));</pre>	3,00	1,60	1.812	- D.D%			0x52d4c1 0x52d4c6	341 341	move Ndx, -0x3m(Nrsp) shr 50x10, Nr10		
	/* B.2 copy 2 mbuf point into rx pkts */							0x52d4ca	341	move Ar10v, -0x2a(Arsp)		
-	_mm_storeu_sil28((ml28i *)℞_pkts[pos], mbpl);		-					= 0x52d4d0	341	nov hrdx, hrlo		
-	Terrare and an and a second se		-					0x52d4d3	341	shr \$0x30, Ardx		
-	/* B 1 load 1 mbuf point */							- 0x52d4d7	341	shr \$0x20, \r10		
	<pre>mbp2 = _mm_loadu_mil20((ml28i *)&sw_ring[pos+2]);</pre>							0x52d4db	341	movy Ndx, -0xa(Nrap)		
		1.00						0x52d4e0	341	novy Arl0v (0xlaiArso)		
	<pre>descs[2] = _mm_loadu_sil28((ml28i *)(rxdp + 2));</pre>	1,40.	600	2.933	2,0%	0.0%	0.000	= 0×52d4e6	347	vmovdgax -0x18(%rsp), %xmmll	0.0% 13.2% 0.0	
	/* B.l load 2 mbuf point */	1.1.1.1						0x52d4ec	348	vmovdqax -0x28(%rsp), %xmml	208 0.4% 0.0% 0.0	
_	<pre>descs[1] = _mm_loadu_sil28((ml28i *)(rxdp + 1));</pre>		200		0.0%				353	vmovdqax -0x40(%rsp), %xmm3	848 0.0% 313% 0.0	00.0 000.0 000
-	descs[0] = _mm_loadu_sil28((ml28i *)(rxdp));	1.00	200,	2.000	0.0%	50.0%			347	upshufb %xmm9, %xmm11, %xmm6		
-	An entry of the state of the state of the state		-				-	0x5264fd	370	vpaddv Nxmn5, Nxmn7, Nxmn5	000 0.0% 0.0% 0.0	1000 1.000 0.00
	<pre>/* B.2 copy 2 mbuf point into rx_pkts */ _mm_stormu_sil28((ml28i *)℞_pkts[pos+2], mbp2);</pre>							0×52/601 0×52/607	353 348	vmovdgax -0x38(%rsp), %xmm4 vpshufb %xmm9, %xmm3	000-0.046-0.046-0.0	00. 1000-000
-	Twificture arrest Tweet claudinestinest, websit							0x52d50c	351	vpurpckhdq %xmal, %xmal, %xmal	Code	
-	if (mplit_packet) {	1.20	2.00	1,200	0.0%	0.0%			355	vpurpckhed \xeell, \xeel, \xeel	coue	
	<pre>rte_prefetch0(℞_pkts(pos)->cachelinel);</pre>	1						0×52d515	355		ing writes]	
	<pre>rte_prefetch0(@rx_pkts[pos + 1]->cachelinel);</pre>							0×523519	372	vpaddy 5xm5, 5xm7, 5xm5	ing writes	
	rte_prefetch0(℞_p&ts[pos + 2]->cachelinel);							0×52d51d	355	vpurpckhvd %xm4, %xm3, %xm2		
	<pre>rte_prefetch0(℞_pkts(pos + 3)->cachelinel);</pre>		-					0x52d521	355	vpunpckldq %xmml, %xmm2, %xmm2		
-)		-					0×52d525	355	vpand %xem2, %xmm14, %xmm2		
-								0×52d529	353	vpunpckhdq %xmm3, %xmm4, %xmm10		_
-	/*shift the pktlen field*/	-	-	-	-	-	-	0x52d52d	355	vpshufb txmm2, txmm13, txmm1		
1	desc_sktlen_align(descs);	-	-	-		-	-	0x52d532 0x52d537	355	vpsrlv \$0xc, %xmm2, %xmm2 vpshufb %xmm9, %xmm4		
1	/* avoid compiler reorder optimization */							0x52d53c	367	vpurpckldg %xmm0, %xmm10, %xmm0		
	rte_compiler_barrier();							0x52d540	355	vpshufb %xms2, %xms12, %xms2		
								0x52d545	355	vpor Axmn2, Axmn1, Axmn2		
	/* D.1 pkt 3,4 convert format from desc to pktmbuf */							0x52d549	355	veovg Sxee2, Srdx		
	pkt_mb4 = _mm_shuffle_epi8(descs[3], shuf_msk);	1.00	2.60		0.0%	1 mm mail	0.000	0x52d54e	363	vpshufb hxmm9, hxmm3, hxmm3		



The Source of the problem

- When assigning the lengths, we use 16-bit writes:
 - have to go to memory (can't assign to an xmm register)
 - causes the xmm load to immediately store to stack
 - causes the shuffle op to trigger a second load
- That second load (128b) blocks on 16b write

```
pktlen0 = _mm_srli_epi32(pktlen0, PKTLEN_SHIFT);
pktlen0 = _mm_and_si128(pktlen0, pktlen_msk);
```

```
pktlen0 = _mm_packs_epi32(pktlen0, zero);
vol.dword = mm cvtsi128 si64(pktlen0);
```

```
/* let the descriptor byte 15-14 store the pkt len */
*((uint16_t *)&descs[0]+7) = vol.e[0];
*((uint16_t *)&descs[1]+7) = vol.e[1];
*((uint16_t *)&descs[2]+7) = vol.e[2];
*((uint16_t *)&descs[3]+7) = vol.e[3];
```



The Fix

Rewrite to use entirely vector operations:

- keeps things in xmm registers
- saves unnecessary loads and stores
- prevents store forward errors
- improves performance.
- makes people happy*

*NOTE: happiness not guaranteed

```
pktlen0 = _mm_srli_epi32(pktlen0, PKTLEN_SHIFT);
pktlen0 = _mm_and_si128(pktlen0, pktlen_msk);
pktlen0 = _mm_packs_epi32(pktlen0, pktlen0);
descs[3] = _mm_blend_epi16(descs[3], pktlen0, 0x80);
```

```
pktlen0 = _mm_slli_epi64(pktlen0, 16);
descs[2] = _mm_blend_epi16(descs[2], pktlen0, 0x80);
```

```
pktlen0 = _mm_slli_epi64(pktlen0, 16);
```

```
descs[1] = _mm_blend_epi16(descs[1], pktlen0, 0x80);
```

```
pktlen0 = _mm_slli_epi64(pktlen0, 16);
```

```
descs[0] = _mm_blend_epi16(descs[0], pktlen0, 0x80);
```



Result

🛅 😥 🕨 😺 🚳 📽 🕐 wices	e rillige X
neral Exploration General Explora	tion viewpoint (change) @
Analysis Target 🚔 Analysis Type 🔳 Collection Log	a 😩 Summary 📣 Battomup 📣 Event Court 📴 Flatform
Elapsed Time [©] : 0.069s	
Clockticks:	130,000,000
Instructions Petired:	182,800,000
CPI Bate ^O t	0.711
MUX Reliability 7:	0.754
Front-End Bound	0.8%
Bad Speculation	0.4%
Back-End Bound	73.8%
describe a portion of the pipeline when	ered due to a lack of required resources for accepting more uOps in the back-end of the pipeline. Back-end metrics e the out-of-order scheduler dispatches ready uOps into their respective execution units, and, once completed, these a order. Stalls due to data-cache misses or stalls due to the overloaded divider unit are examples of back-end bound
Memory Bound	50.7%
	dicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores. ccess analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by
certain cases like loads blocked	0.215 achine was stalled without missing the L1 data cache. The L1 cache typically has the shortest latency. However, in on older stores, a load might suffer a high latency even though it is being satisfied by the L1. Note that this metric DTLB Overhead or Cycles of 1 Port Utilized issues.
OTLB Overhead [©] :	0.000
Loads Blocked by Store Forwa	rding [®] : 0.010
Split Loads Ct	0.000
4x Alasing the	0.167
	cles is spent dealing with false. 4k aliasing between loads and stores. Use the source/assembly view to identify the I then adjust your data layout so that the loads and stores no longer alias.
FB FUI TI	1000
L2 Bound ^(b) :	0.062
This metric shows how often m performance.	achine was stalled on L2 cache. Avoiding cache misses (L1 misses/L2 hits) will improve the latency and increase
L3 Bound [®] :	0.123
Contested Accesses (*)	0.000
Data Sharing ®:	0.000
L3 Latency (1)	0.886
This metric shows a fraction	of cycles with demand load accesses that hit the L3 cache under unloaded scenarios (possibly L3 latency limited).

Avoiding private cache misses (i.e. L2 misses[L3 hts) will improve the latency, reduce contention with sibling physical cores and increase performance. Note the value of this node may overlap with its siblings.

- Stalls due to Loads Blocked by Store Forwarding dropped about 19x:
 - Before: 0.189
 - After: 0.010
- Overall PMD performance measured by testpmd increased by over 5%.

