Extending Performance Monitoring Profile Guided Optimization Capabilities

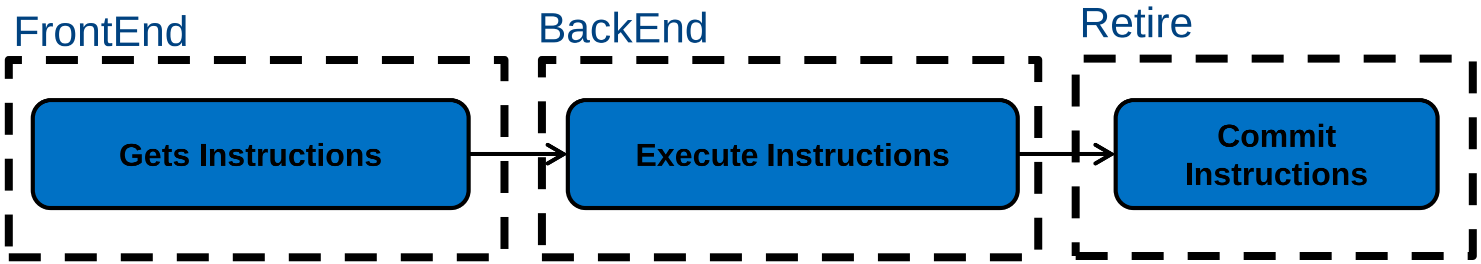
Michael Chynoweth - Sr. Principal Engineer Intel Corporation

Contributors: Joe Olivas, Chris Chrulski, Patrick Konsor, Rajshree Chabukswar, Stas Bratanov, Hideki Saito, Angie Schmid, Sneha Gohad, Robert Cox, Zia Ansari, Ahmad Yasin, Lama Saba, Dorit Nuzman

Agenda

- Today Profile Guided Optimizations are mostly impacting code/text section
  - Extensions on analysis to the text section optimizations
- Who’s Interested?
- Next generation of PGO will utilize more events
  - Allow focus on the right bottleneck
- Examples of automatic profile guided optimizations with compiler
  - Decision on whether to fix a uarch bottleneck
  - Loop optimizations
  - Data reordering

Top Down: Our Processor is Just An Assembly Line



- Abstracts our architectures into 4 categories
  - Front End Bound
  - Back End Bound
  - Bad Speculation
  - Retiring
- Focus our efforts on the right bottlenecks

Top Down Helps Define the Primary Bottleneck

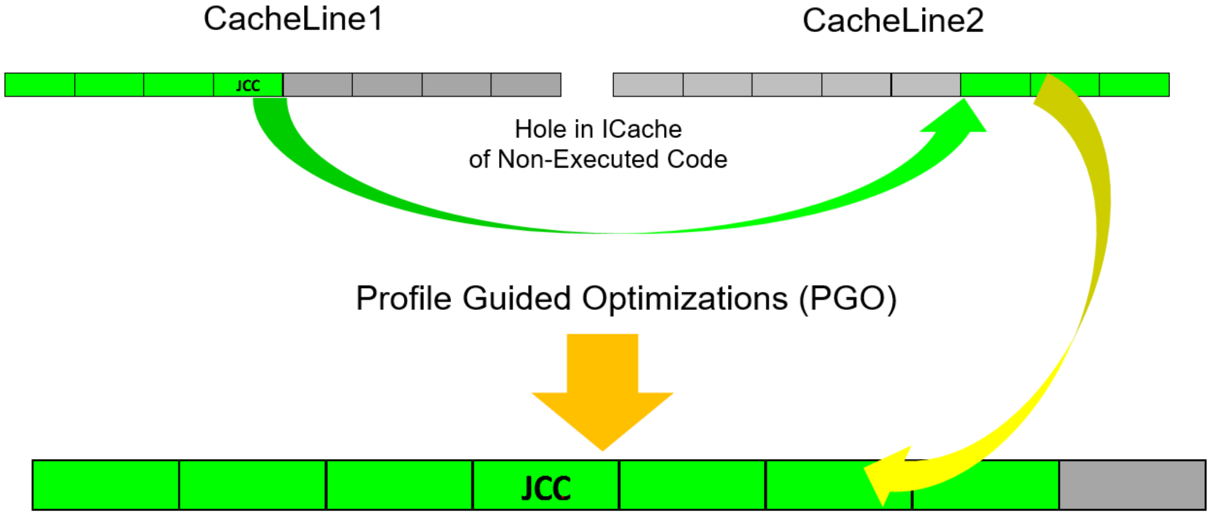
Everything is Driven by Top Down Optimizations

Metric	Cost	Performance Monitoring Events Calculation
Front End Bound Cost	**38.8%**	**NO_ALLOC_CYCLES.NOT_DELIVERED/CPU_CLK_UNHALTED.CORE**
Instruction Cache Misses Cost	**26.3%**	INST_LINE_FETCH_COST+PREDECODE_WRONG_COST
Instruction Line Fetch Cost	7.2%	FETCH_STALL.ICACHE_FILL_PENDING_CYCLES*1/CPU_CLK_UNHALTED.CORE
PreDecode Wrong Cost	19.1%	DECODE_RESTRICTION.PDCACHE_WRONG*3/CPU_CLK_UNHALTED.CORE
ITLB Misses Cost	**8.5%**	PAGE_WALKS.I_SIDE_CYCLES*1/CPU_CLK_UNHALTED.CORE
Back End Bound Cost	**44.1%**	**1-RETIRING-FRONT_END_BOUND-BAD_SPECULATION**
L2 Data Miss Cost	12.0%	MEM_UOPS_RETIRED.L2_MISS_LOADS_PS*230/CPU_CLK_UNHALTED.CORE
DTLB Misses Cost	**9.0%**	PAGE_WALKS.D_SIDE_CYCLES*1/CPU_CLK_UNHALTED.CORE
Bad Speculation Bound Cost	**3.6%**	**NO_ALLOC_CYCLES.MISPREDICTS*1/CPU_CLK_UNHALTED.CORE**
Branch Mispredict Cost	5.70%	BR_MISP_RETIRED.ALL_BRANCHES_PS*10/CPU_CLK_UNHALTED.CORE
Retiring Bound Cost	**13.5%**	**UOPS_RETIRED.ALL*0.5/CPU_CLK_UNHALTED.CORE**

Fixed issues in red… will cover later

Performance Monitoring Tells Where We are Bound and By How Much

PGO Example Basic Block Reordering



Profile Guided Optimizations (PGO) increases efficiency of ICache

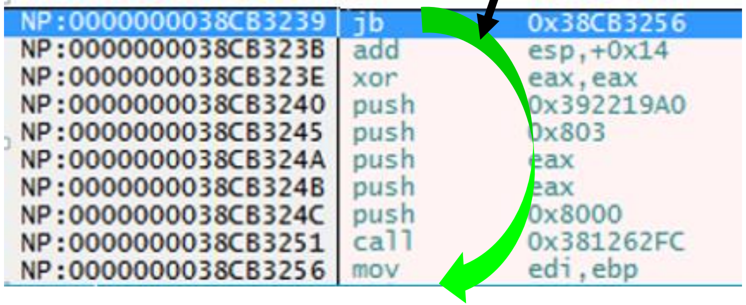
Successful Basic Block Reordering

Statistic	NoPGO	PGO
%FWD_TAKEN_JCC	31%	16%

Unsuccessful Basic Block Reordering

Statistic	NoPGO	PGO
%FWD_TAKEN_JCC	28%	29%

Jumps over debug code 100% time



*%FWD_TAKEN_JCC = (FWD_TAKEN_JCC-FWD_TAKEN_JCC_LESSTHAN_10BYTES)*100/ALL_CONDITIONAL*

LBR Already Gives Us Overall Statistics Allowing Prediction of Opportunity

Predicted using LBR

PotentialInstructionCacheSavedPercentage	11.6%
BranchWith4kTraversalPercentage	36.3%

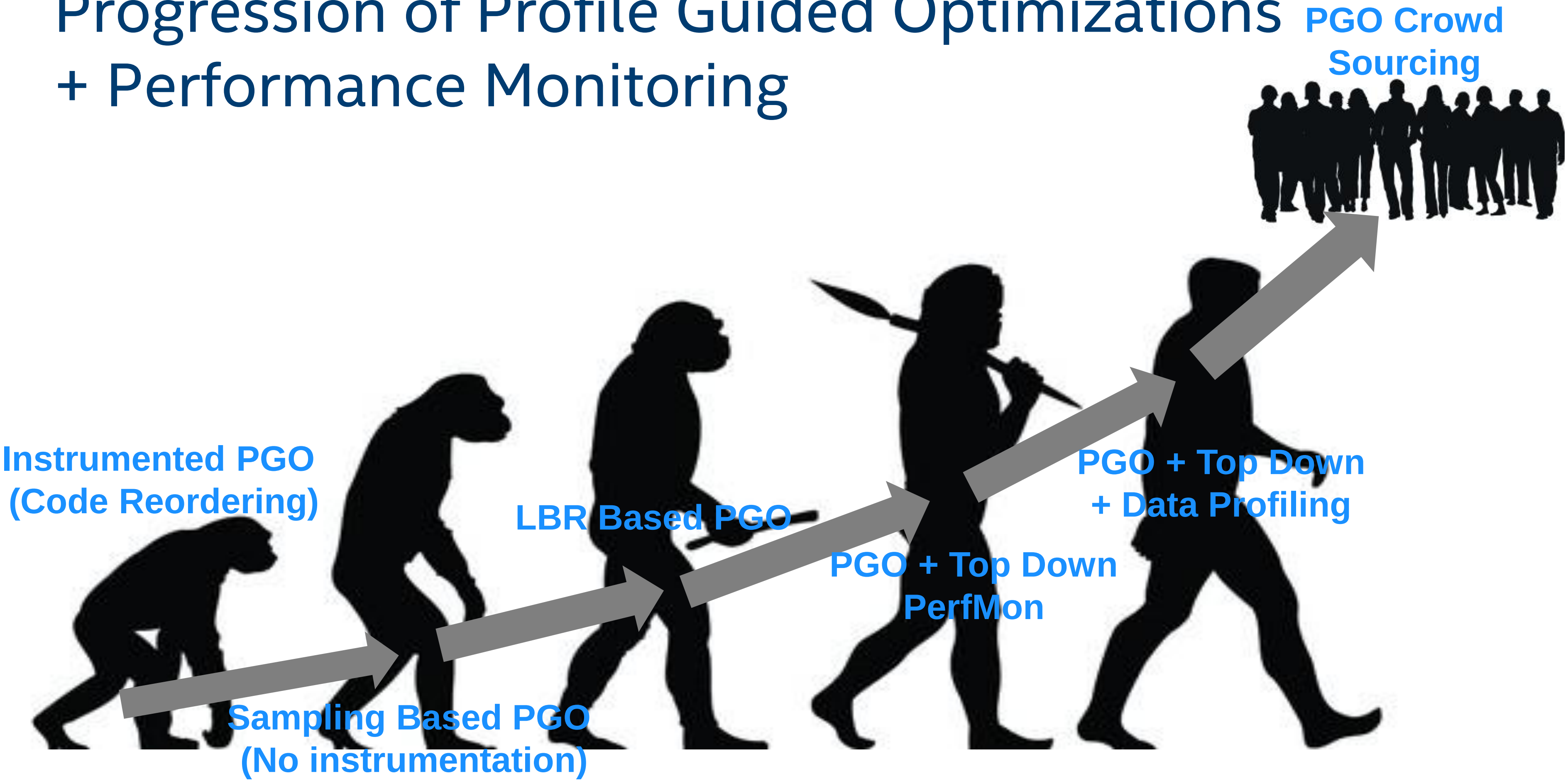
Statistic	NoPGO	PGO
TotalBytesExecuted	69k	62k
TotalCacheLinesExecuted	1738	1373
TotalCacheLinesBytes	109k	86k
CacheLineEfficiency	64%	72%
TotalPagesExecuted	182	93
PageEfficiency	10%	17%

Statistic	No PGO	PGO	PGO/NoPGO
Utilization:	39%	33%	1.18
Front End Bound Cost	43%	32%	

Taking Profile Guided Optimizations to Next Level

- Utilize all of performance monitoring capabilities for PGO
- Code reorganization (Already being stressed)
  - Basic block + Function reordering, Function splitting, Inlining/partial inlining
- Data profiling
  - Data structure + Data section reordering + False sharing avoidance
  - Function parameters
  - Loop pointer aliasing
  - Intelligent allocators
- Drive optimizations based on where bound in the pipeline
  - Often optimizations conflict
    - Example = "optimize for speed" and "optimize for size"
  - Loop vectorization
  - Fixing individual code generation issues

Progression of Profile Guided Optimizations + Performance Monitoring



Top Down Helps Determine Usage of Compiler Workaround for Slow LEA (LLVM Compiler)

Issue Type	Assembly
SLOW_LEA	lea rax,ptr [r9+rax*1-fff1]

Slower execution

Statistics	SlowLEA	SlowLEA Patch	SlowLEA/ SlowLEAPatch
Benchmark Cycles Per Instruction (CPI)	0.60	0.59	1.03
Benchmark Front End Bound Cost	9.4%	10.2%	0.92
Benchmark Core Bound	22.1%	17.2%	1.28
Benchmark Slow LEA	**5.7%**	**2.4%**	**2.38**

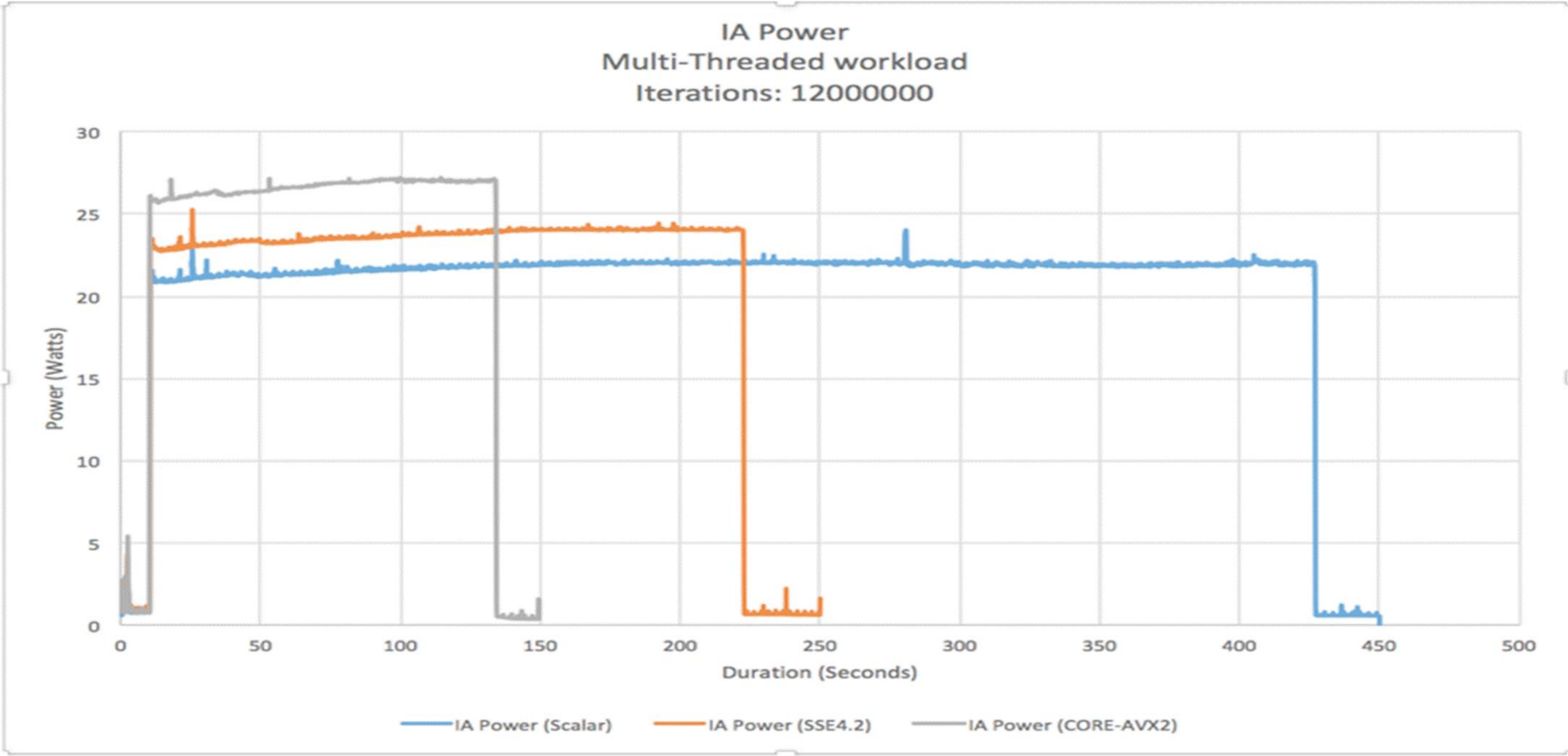
Front end bottleneck increases

Core bound cost due to slow lea decreases

How Can Performance Monitoring PGO Help Optimize a Loop?

- Picked a couple of examples loops from benchmarks to create proof-of-concepts
- Loops were unique in that we could force them to auto-vectorize with pragmas
  - Gave us 2.6% speedup on the benchmark (on ICC or LLVM)
- Information could Performance Monitoring for PGO Provide?
  - % Cost of loop within process
    - Determines how aggressive to attempt vectorize
  - Average trip count of loop
  - Typical values in the loop
    - A value of shift in the loop is always zero
  - Pointer aliasing and data alignment
  - Total time in all vectorizable loops in the process

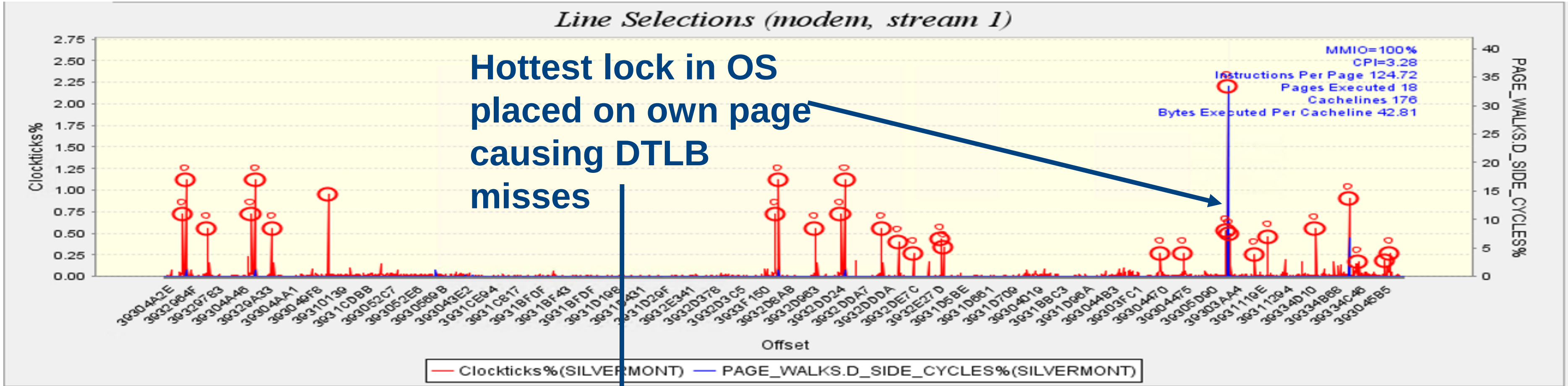
Choosing Which Level of Vectorization to Utilize



Top Down and Data Reordering

Metric	PGO	PGO + Full Interprocedural Opt
Back End Bound Cost	43%	49%
DTLB Misses Cost	1%	8%

Compiler optimization Hurting performance Due to data locality



Hottest lock in OS placed on own page causing DTLB misses

Selection Granularity

Instr Line

Export

ST#	HB#	BB#	LN#	Offset	Opcode	Disasm	Latency(SILVERMONT)	Static ASM Notes	Function
1	161	0	2	39334C06	8C E1	mov ecx, fs	1		
1	161	0	3	39334C08	39 0D 04 0C…	cmp dword ptr [3b720c04], ecx	1	GLOBAL_ACCESS	
1	161	0	4	39334C0E	74 0F	jz 39334c1f	1		
1	161	1	0	39334C10	B8 FF FF FF FF	mov eax, ffffffff	1		
1	161	2	0	39334C15	F0 0F B1 0D…	lock cmpxchg dword ptr [3b720c04], ecx		ATOMIC_INSTRUCTION:GLOBAL_ACCESS	_retry_protection

Top Down Helps Identify Necessary Global Data Reordering

Conclusions

- Today Profile Guided Optimizations (PGO) mostly impacting code/text section
  - Easier than impacting other vectors
- Next generation of PGO will utilize more events and capabilities
  - Determine where the instruction pipeline is bound
  - Appropriately address the appropriate bottleneck
  - Currently taking advantage of a small portion of opportunity
- Started an effort to tackle
  - Covered uarch optimization, loop optimizations and data reorganization

Backup