Performance Analysis and Loop Optimization

David Levinthal
Principal Engineer
SSG/DPD

Legal Disclaimer

Risk Factors

This presentation contains forward-looking statements that involve a number of risks and uncertainties. These statements do not reflect the potential impact of any mergers, acquisitions, divestitures, investments or other similar transactions that may be completed in the future. The information presented is accurate only as of today's date and will not be updated. In addition to any factors discussed in the presentation, the important factors that could cause actual results to differ materially include the following: Demand could be different from Intel's expectations due to factors including changes in business and economic conditions, including conditions in the credit market that could affect consumer confidence; customer acceptance of Intel's and competitors' products; changes in customer order patterns, including order cancellations; and changes in the level of inventory at customers. Intel's results could be affected by the timing of closing of acquisitions and divestitures. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of new Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel's response to such actions; Intel's ability to respond quickly to technological developments and to incorporate new features into its products; and the availability of sufficient supply of components from suppliers to meet demand. The gross margin percentage could vary significantly from expectations based on changes in revenue levels; product mix and pricing; capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; excess or obsolete inventory; manufacturing yields; changes in unit costs; impairments of long-lived assets, including manufacturing, assembly/test and intangible assets; and the timing and execution of the manufacturing ramp and associated costs, including start-up costs. Expenses, particularly certain marketing and compensation expenses, vary depending on the level of demand for Intel's products, the level of revenue and profits, and impairments of long-lived assets. Intel is in the midst of a structure and efficiency program that is resulting in several actions that could have an impact on expected expense levels and gross margin. Intel's results could be impacted by adverse economic, social, political and physical/infrastructure conditions in the countries in which Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. A detailed discussion of these and other factors that could affect Intel's results is included in Intel's SEC filings, including the report on Form 10-Q for the quarter ended June 28, 2008.

Agenda

- Dominant issues in loop analysis
- Using LBRs
- Example

What matters when optimizing a loop?

1. The Trip Count
2. The Trip Count
3. The TRIP COUNT!
4. Variations in the tripcount
5. And some other things

BUT..what you do about them depends on

THE TRIP COUNT

And of course there are virtually no tools to assist you in determining this..other than printf

(you can use PIN..)

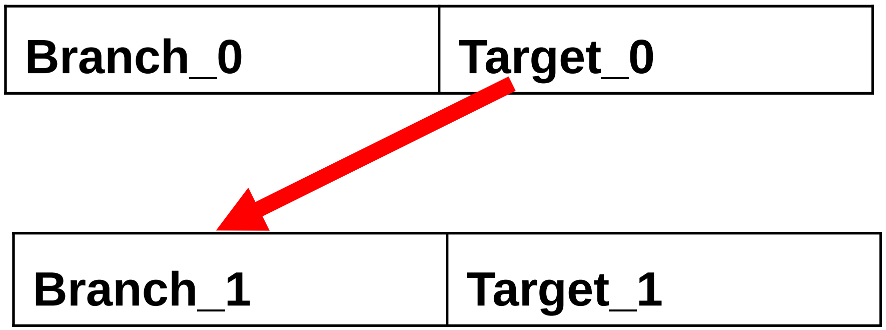
Basic loop tripcount ranges (for short loops)

- The tripcount dictates optimization options, as it defines the time available to amortize the cost of the proposed solution
- Short loops tripcount < 7?
  - Unroll the loop completely/vectorize
- Medium loops 7<tripcount<15-20?
  - Most difficult case..too short to do much of anything, only option is vectorize
- Medium_long   20<tripcount<50
  - Almost long enough for real options
- Long loops 50-100 < tripcount
  - Lots of options
- Of course all loops have their own issues

Basic Branch Analysis

- **Vastly improved precise branch monitoring capabilities**
  - **16 deep Last Branch Record (LBR)**
    - Records <u>Taken</u> Branches and their targets
    - LBR can be filtered by branch type and privilege level
- **Precise br retired by branch type**
  - **Calls, conditional and all calls**
  - **Coupled with LBR capture yields**
    - Call counts
    - Basic Block execution counts
    - “HW call graph”

Processing LBRs



- All instructions between Target_0 and Branch_1 are retired 1 time
- All Basic Blocks between Target_0 and Branch_1 are executed 1 time
- All Branch Instructions between Target_0 and Branch_1 are not taken

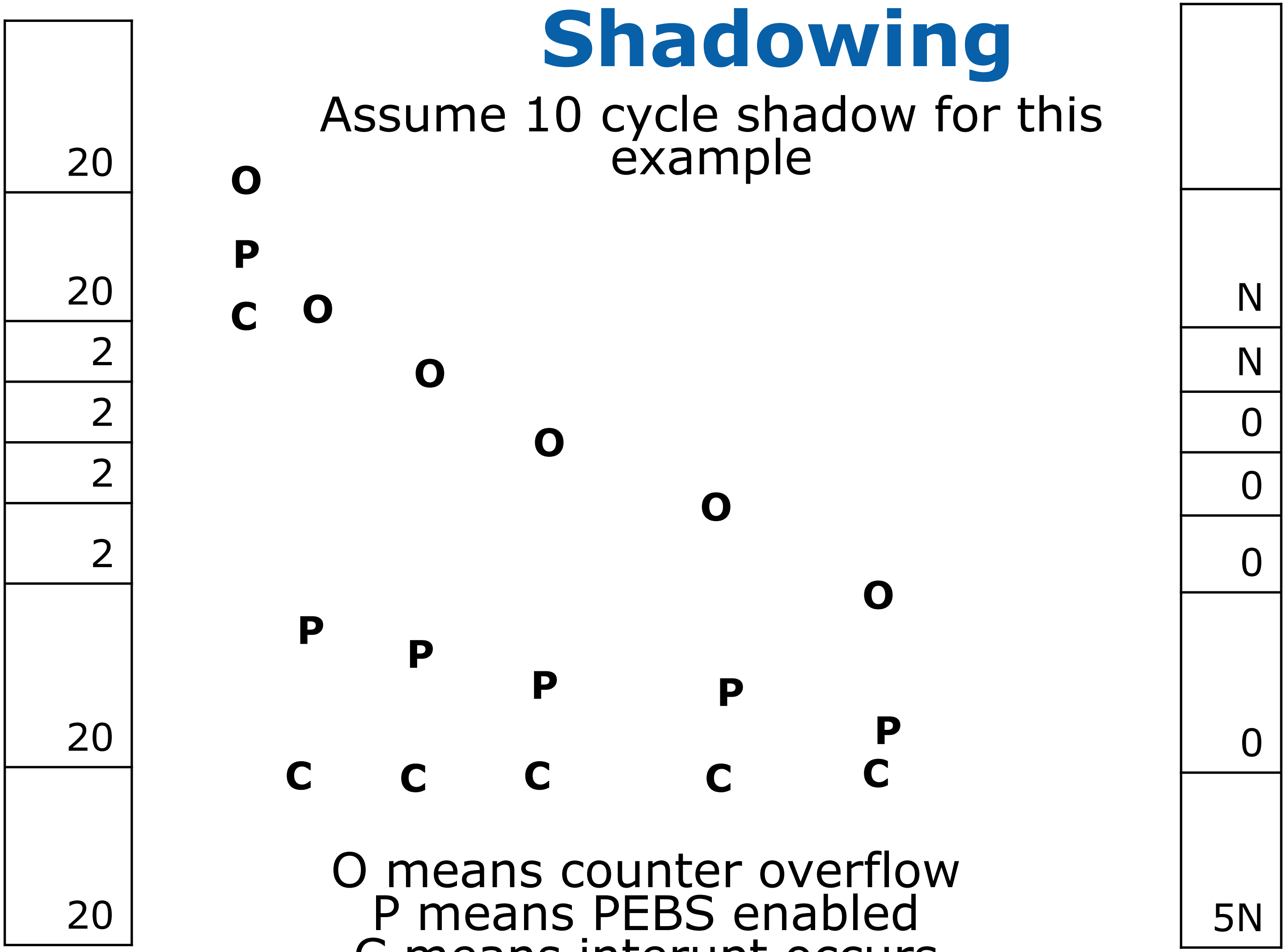
So it would all Seem Very Straight Forward

Shadowing and Precise Data Collection

- The time between the counter overflow and the PEBS arming creates a "shadow", during which events cannot be collected
  ~8 cycles?
- Ex: conditional branches retired
  - Sequence of short BBs (< 3 cycles in duration)
  - If branch into first overflows counter, Pebs event cannot occur until branch at end of 4th BB
  - Intervening branches will never be sampled

Shadowing

Assume 10 cycle shadow for this example



O means counter overflow
P means PEBS enabled
C means interupt occurs

Reducing Shadowing Impact

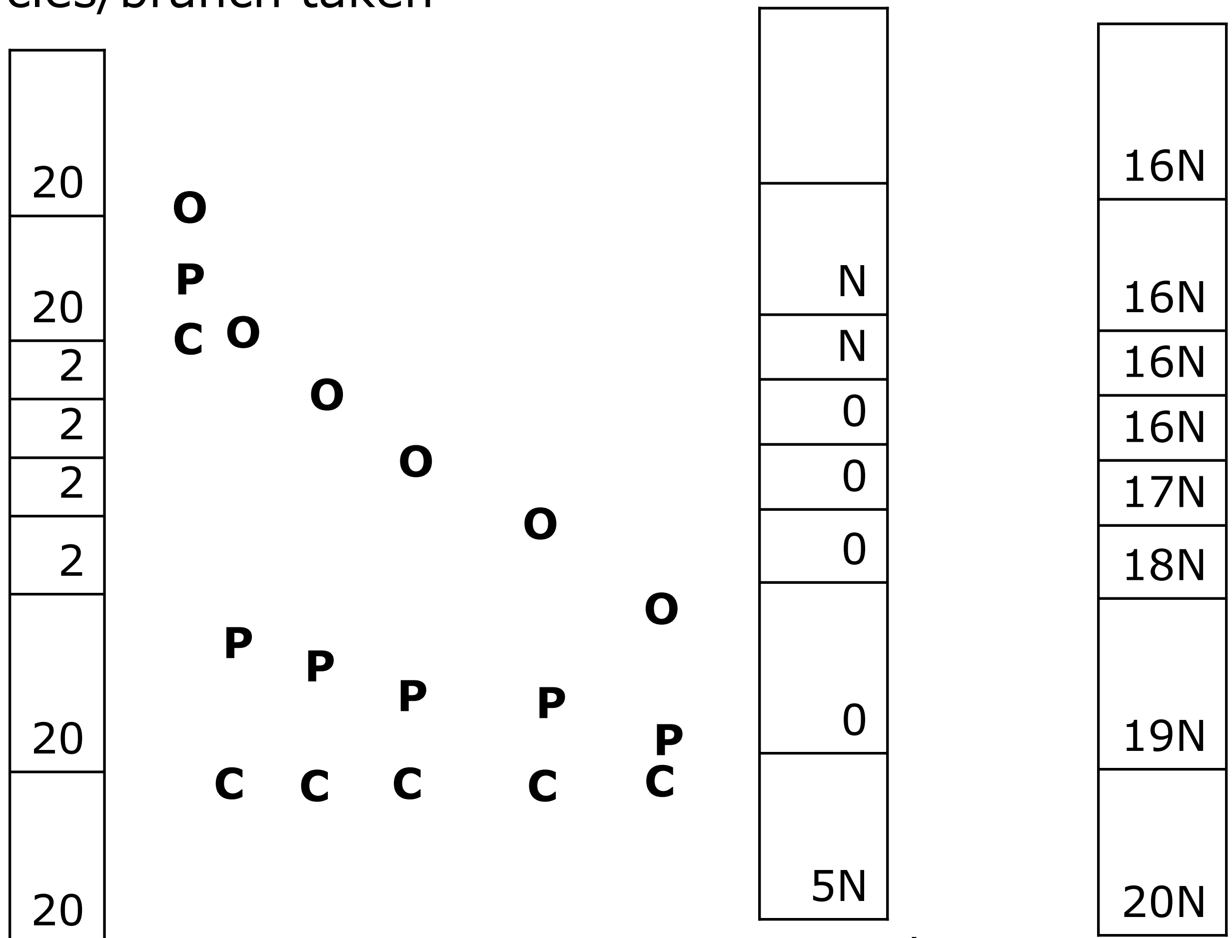
- Some "events" will never occur!
  - Falling into shadowed window
- Use LBR to extend range of the single sample
- Count the number of objects in LBR and increment count for all of them by 1/NUM
  - Since you have only one sample

Minimizing Shadowing Impact on BB Execution Count

Cycles/branch taken

Pebs Samples taken

Number of LBR entries



Many more with 20 Cycles/branch taken

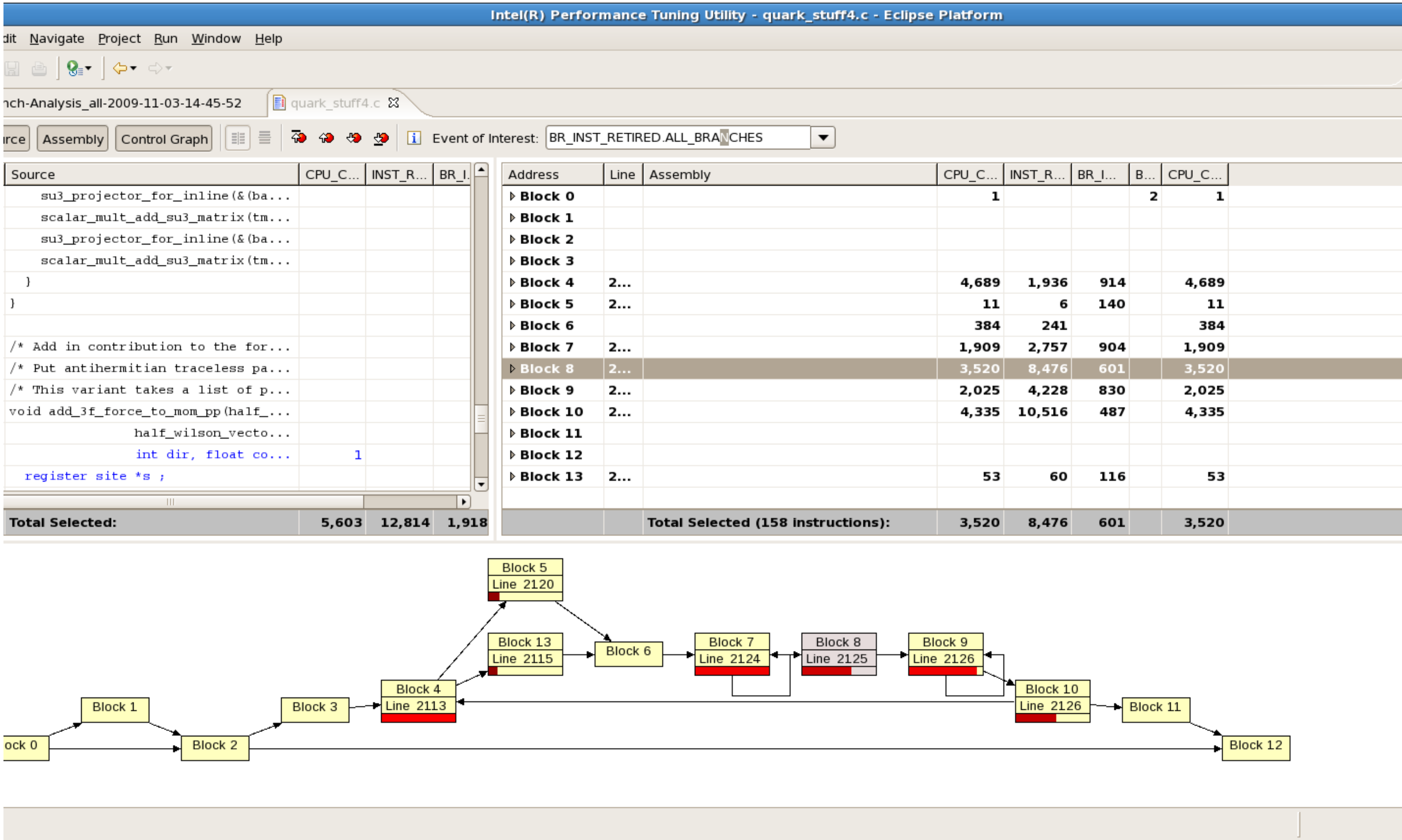
Many more with N samples taken

Many more with 16 N LBR Entries

In this example there are always 16 BB's covered in the LBR.

Incrementing the BB execution count for each BB detected in the LBR, by 1/NUM_LBR-1 will greatly reduce the effect of shadowing

Nested Loop with 8 Basic Blocks



Source	CPU_C...	INST_R...	BR_I...
su3_projector_for_inline(&(ba...			
scalar_mult_add_su3_matrix(tm...			
su3_projector_for_inline(&(ba...			
scalar_mult_add_su3_matrix(tm...			
}			
}			
/* Add in contribution to the for...			
/* Put antihermitian traceless pa...			
/* This variant takes a list of p...			
void add_3f_force_to_mom_pp(half_...			
half_wilson_vecto...			
int dir, float co...	1		
register site *s ;			
Total Selected:	**5,603**	**12,814**	**1,918**

Address	Line	Assembly	CPU_C...	INST_R...	BR_I...	B...	CPU_C...
Block 0			1			2	1
Block 1							
Block 2							
Block 3							
Block 4	2...		4,689	1,936	914		4,689
Block 5	2...		11	6	140		11
Block 6			384	241			384
Block 7	2...		1,909	2,757	904		1,909
Block 8	2...		3,520	8,476	601		3,520
Block 9	2...		2,025	4,228	830		2,025
Block 10	2...		4,335	10,516	487		4,335
Block 11							
Block 12							
Block 13	2...		53	60	116		53
		Total Selected (158 instructions):	**3,520**	**8,476**	**601**		**3,520**

Basic Block Execution Counts

Basic Block	Instructions	inst_ret	BB_exec(inst)	br_ret	lbr_all	lbr_tkn	expected
0	4	0	0	0	0	0	
1	8	0	0	0	0	0	
2	6	0	0	0	0	0	
3	8	0	0	0	0	0	
4	3	1036	345.33	914	9267	6286	1
5	2	6	3	140	5054	3167	0.5
6	11	241	21.91	0	9270	6287	1
7	40	2757	68.92	904	28312	19394	3
8	158	8476	53.65	601	9543	6726	1
9	40	4228	105.7	830	28602	19761	3
10	183	10516	57.46	487	9633	6727	1
11	3	0	0	0	5	2	
12	2	0	0	0	5	2	
13	5	60	12	116	4251	3122	0.5
					3990	2762	
tripcount			78.793	405.1429	9427.048	6480.952	
sav			2000000	200000	13333.3	13333.3	
loop executions			157587619	81028571	1.26E+08	86412482	

Normalization = Tripcount

- Using br_inst_retired.near_call is difficult as the function is only called a few thousand times, while the hottest function is called ~ 1million times
- The true tripcount is 16384
- “call count” = loop_executions/16384

BB_exec(inst)	br_ret	lbr_all	lbr_tkn
9618.38	4945.59	7671.73	5274.2

True Call Count from printf ~ 5750
(done with new 11.1 based build)

Summary

- LBRs are very useful