インテル® Core™ マイクロアーキテクチャー・プロセッサーのパフォーマンス・カウンター

ソフトウェア＆ソリューションズ統括部

ソフトウェア製品部

インテル® VTune™ パフォーマンス・アナライザー 9.0

発見が困難なパフォーマンスのボトルネックを識別

機能

- プロセスまたはスレッド並列コードのチューニング
- オーバーヘッドの少ないサンプリング
- グラフィカルなコールグラフ
- ソースまたはアセンブリーで結果を表示

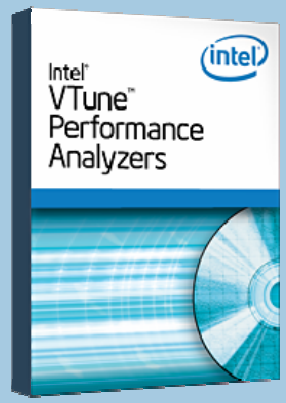
新機能

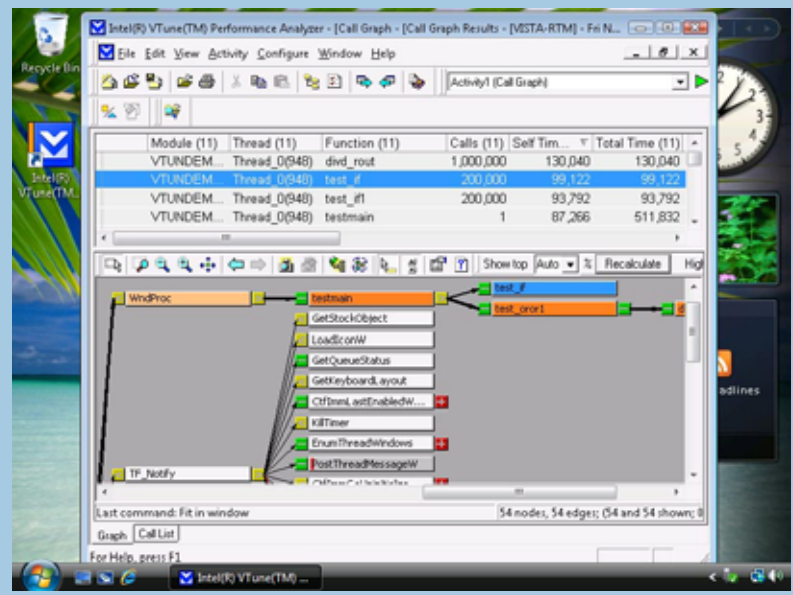
- 新しいチューニング方法論
  - Core™2 Duo プロセッサーと Core™2 Quad プロセッサーのストール・サイクル・アカウンティング
- Windows: Microsoft Vista* 対応
- Linux: インテル® コンパイラーによる分析と直感的な hotspot ナビゲーター

Windows*	Linux*	Mac*	IA32	Intel64	IA64	マルチコア
√	√		√	√	√	√

「インテル® VTune™ パフォーマンス・アナライザーを使えば、これまで何日もかかっていた作業を 1 日以内に完了することができます。」

Randy Camp 氏
ソフトウェア R&D 担当副社長
MUSICMATCH Inc.





インテル® ソフトウェア開発製品

目的

インテル® Core™ マイクロアーキテクチャー・プロセッサー搭載のシステムで実行するソフトウェアのマイクロアーキテクチャー上のボトルネックを識別する

コースの内容

イベントの基本

インテル® Core™ アーキテクチャー・プロセッサーのボトルネックを識別するイベント

まとめ

イベントの基本

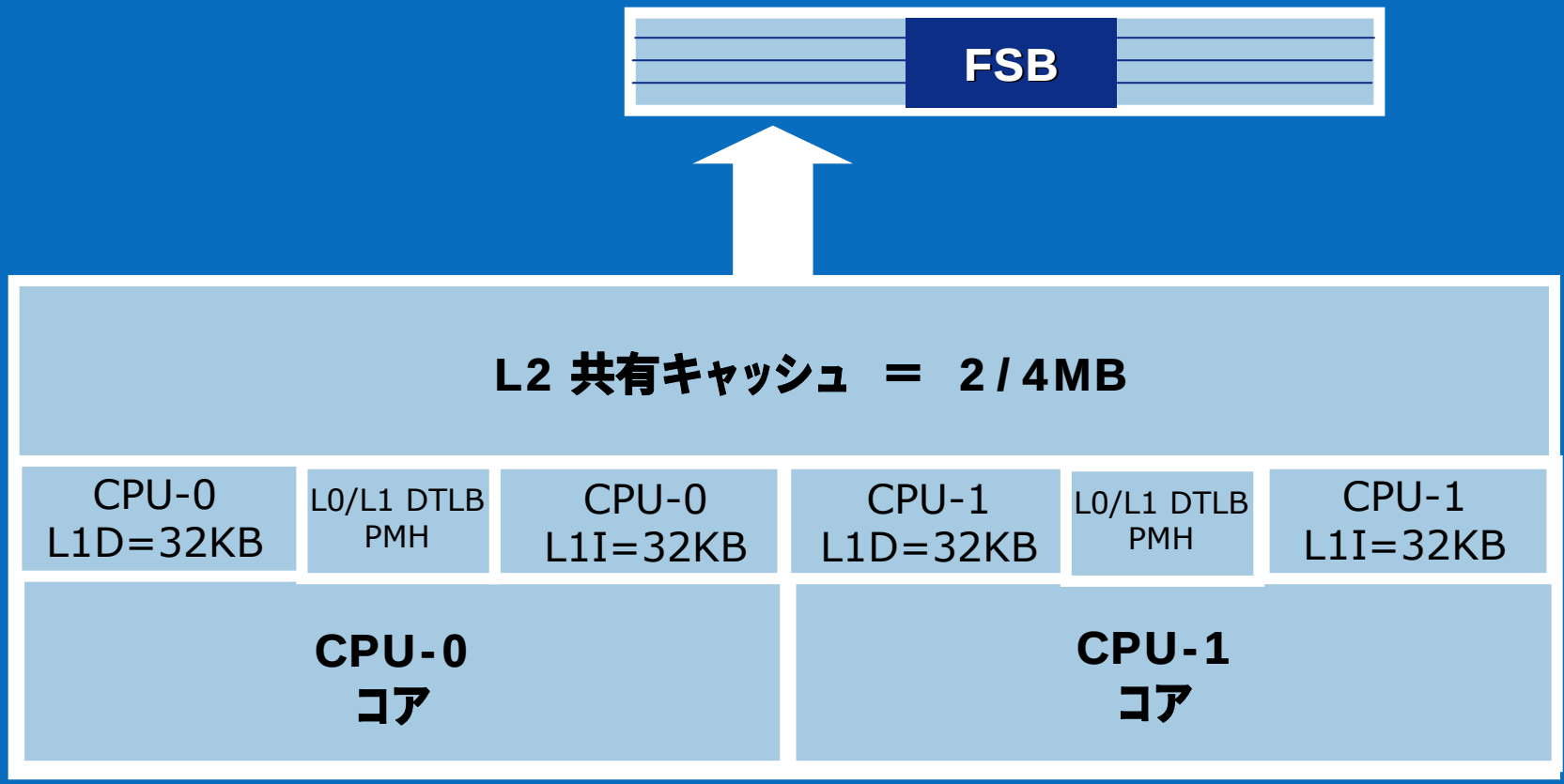
インテル® VTune™ アナライザー:
イベントとサンプル

パフォーマンス・カウンターはプロセッサーでイベントが発生するたびにインクリメントされる

パフォーマンス・カウンターがオーバーフローするごとに、実行結果のサンプルを記録する

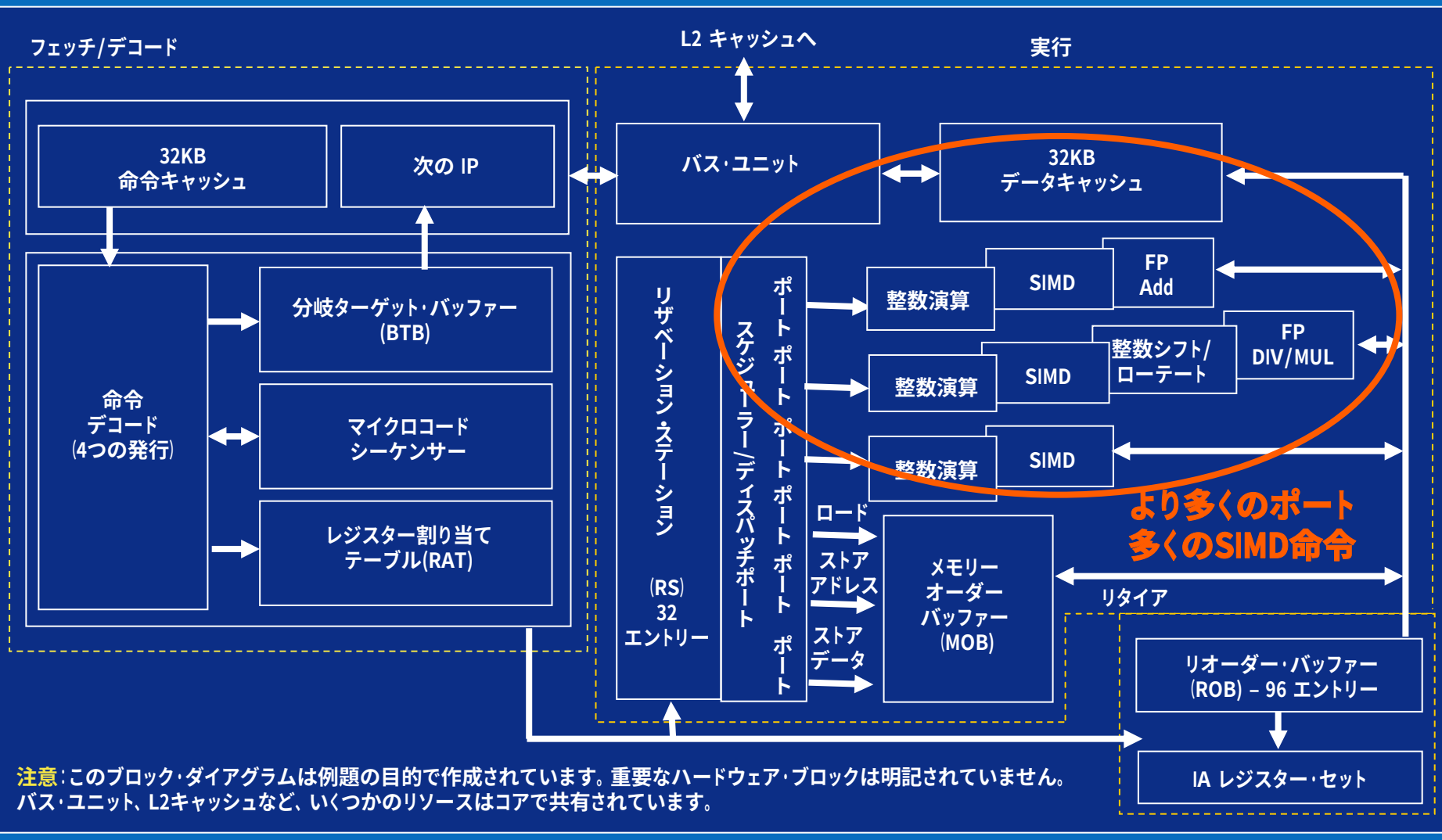
イベント数 = サンプル数 * サンプリング間隔の値 (SAV)

新世代マイクロアーキテクチャー
インテル® Core™ マイクロアーキテクチャー・プロセッサー



インテル® ソフトウェア開発製品

アーキテクチャー・ブロック・ダイアグラム



注意:このブロック・ダイアグラムは例題の目的で作成されています。重要なハードウェア・ブロックは明記されていません。
バス・ユニット、L2キャッシュなど、いくつかのリソースはコアで共有されています。

4 つのハードウェア・プリフェッチャー

- L1 キャッシュ・プリフェッチ
  (インテル® Core™ Duo プロセッサーで最初に実装された)
  - DCU とストリーミング・プリフェッチャー
    - DCU = Data Cache Unit
  - IP プリフェッチャー
    - 頻繁に実行される命令ポインター (IP) における繰り返されるストライド・ロード
- L2 キャッシュ・プリフェッチ
  (インテル® Pentium® 4 プロセッサーと同様)

基本イベント

イベント	P	説明	イベント	P	説明
CPU_CLK_UNHALTED			BUS_DRDY_CLOCKS.ALL_AGENTS		全てのBUSYバス・サイクル
INST_RETIRED_ANY_P	P		BUS_DRDY_CLOCKS.THIS_AGENT		書き込みによる全てのBUSYバス・サイクル
INST_RETIRED_LOADS			MEM_LOAD_RETIRED.L2_LINE_MISS	P	L2の要求ミス
INST_RETIRED_STORES			MMX2_PRE_MISS.T1		L1キャッシュへのSWプリフェッチ
BUS_TRANS_ANY		全てのバス・トランザクション	MMX2_PRE_MISS.T2		L2キャッシュへのSWプリフェッチ
BUS_TRANS_MEM		メモリーへのバス・トランザクション	MMX2_PRE_MISS.STORES		実行された非テンポラルストア
BUS_TRANS_BURST		ラインのメモリー書き込み	L2_LINES_IN.SELF.DEMAND		SWプリフェッチによるL2へのライン転送
BUS_TRANS_BRD		ラインのメモリーからの読み込み	L2_LINES_IN.SELF.PREFETCH		HWプリフェッチによるL2へのライン転送
BUS_TRANS_WB		ライトバック (NTライトはなし)	L2_LINES_OUT.SELF.DEMAND		要求されたL2の追い出し
BUS_TRANS_RFO		RFOへのライン転送 (HWプリフェッチではない)	L2_LINES_OUT.SELF.PREFETCH		HWプリフェッチによるL2の追い出し

メモリーバンド幅 = 64 * Bus_Trans_Mem * 周波数 / Cpu_Clk_Unhalted

方法論の概要

- X86 プロセッサーにおける伝統的なパフォーマンス・チューニングは、命令のリタイアに注目する
- OOO エンジンの実行動作は不透過で、予測することが困難である

そのため命令のリタイアに注目することは最善の方法ではない

方法論の概要

- 方法論は浮動小数点ループ主体のアプリケーションを分析することによって求められた
  - この限られた検証から得られる利益は偶然の産物である
- この方法による最適化は 2 つの要素からなる
  - 命令カウントを最小限にする
    - “木構造”の最小化
  - 理想的な実行ルートからの逸脱を最小限にする
    - 一般的に“ストール・サイクル”として考えられる
  - 両者を同一に扱うことは危険である

ストール、不完全な実行そしてパフォーマンス解析

- ストール・サイクルは実行の不完全さを表す
  - プロセッサーのアーキテクチャーに依存するストールは、アーキテクチャー・イベントを監視することで知ることができる
  - ストールと IP の関連およびアーキテクチャー・イベントは最適化の指標となる
- OOO エンジンには 4 つの代表的なストールがある
  - 実行ストール
    - 入力やスコアボード待ち、L2 ミス、BW、DTLB、その他
  - 分岐予測ミスによるパイプラインのフラッシュ
  - FE (フロント・エンド) ストール
    - 実行ステージで命令が足りなくなる
  - リタイアしない命令によって CPU サイクルが浪費される

X86 サイクル計算とソフトウェア最適化

- CPU_CLK_UNHALTED =
  “ストール” + NON_RET_DISPATCH + RET_DISPATCH

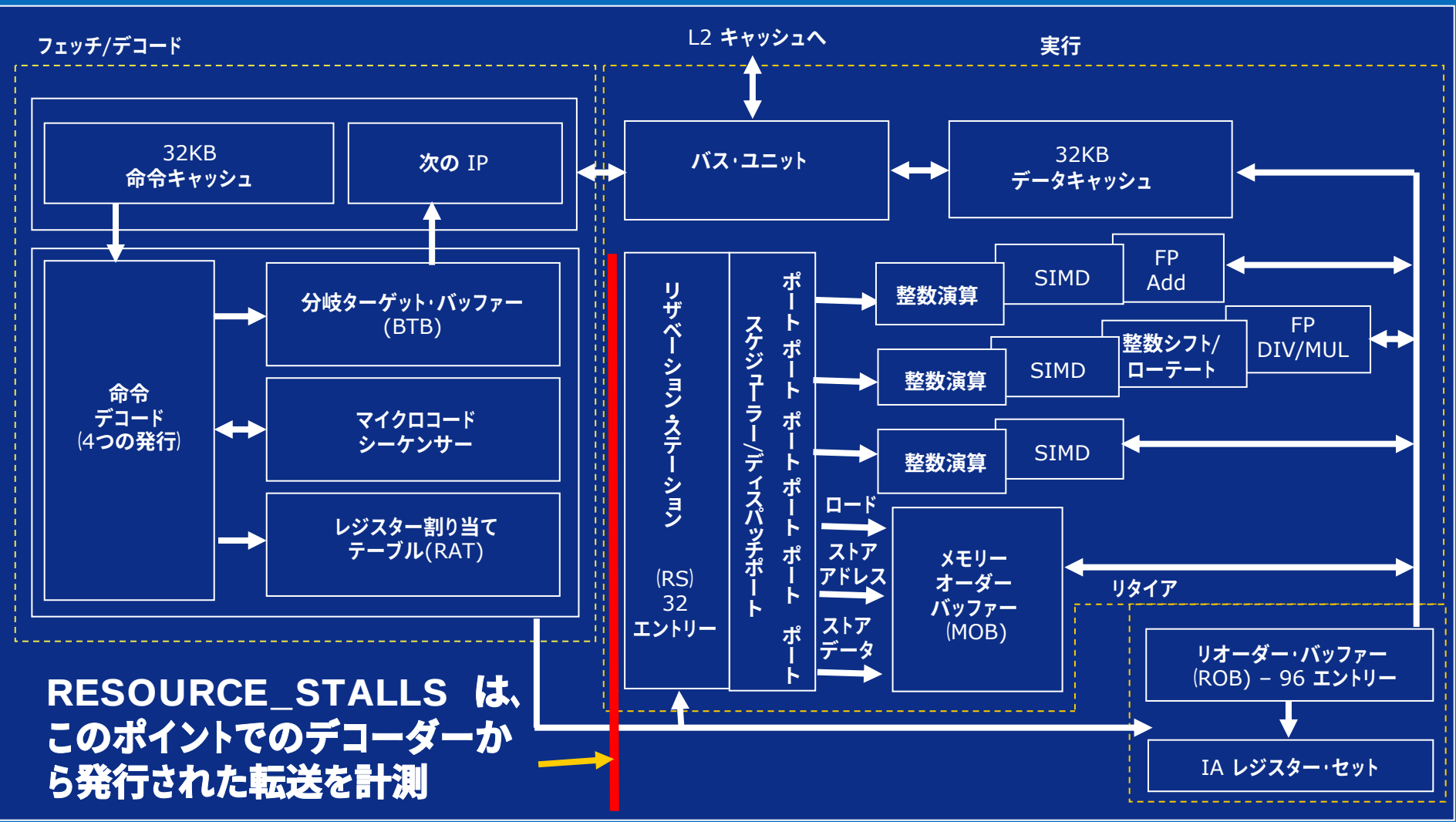
代表的なストールの排除

分岐予測ミスを減らす PGO

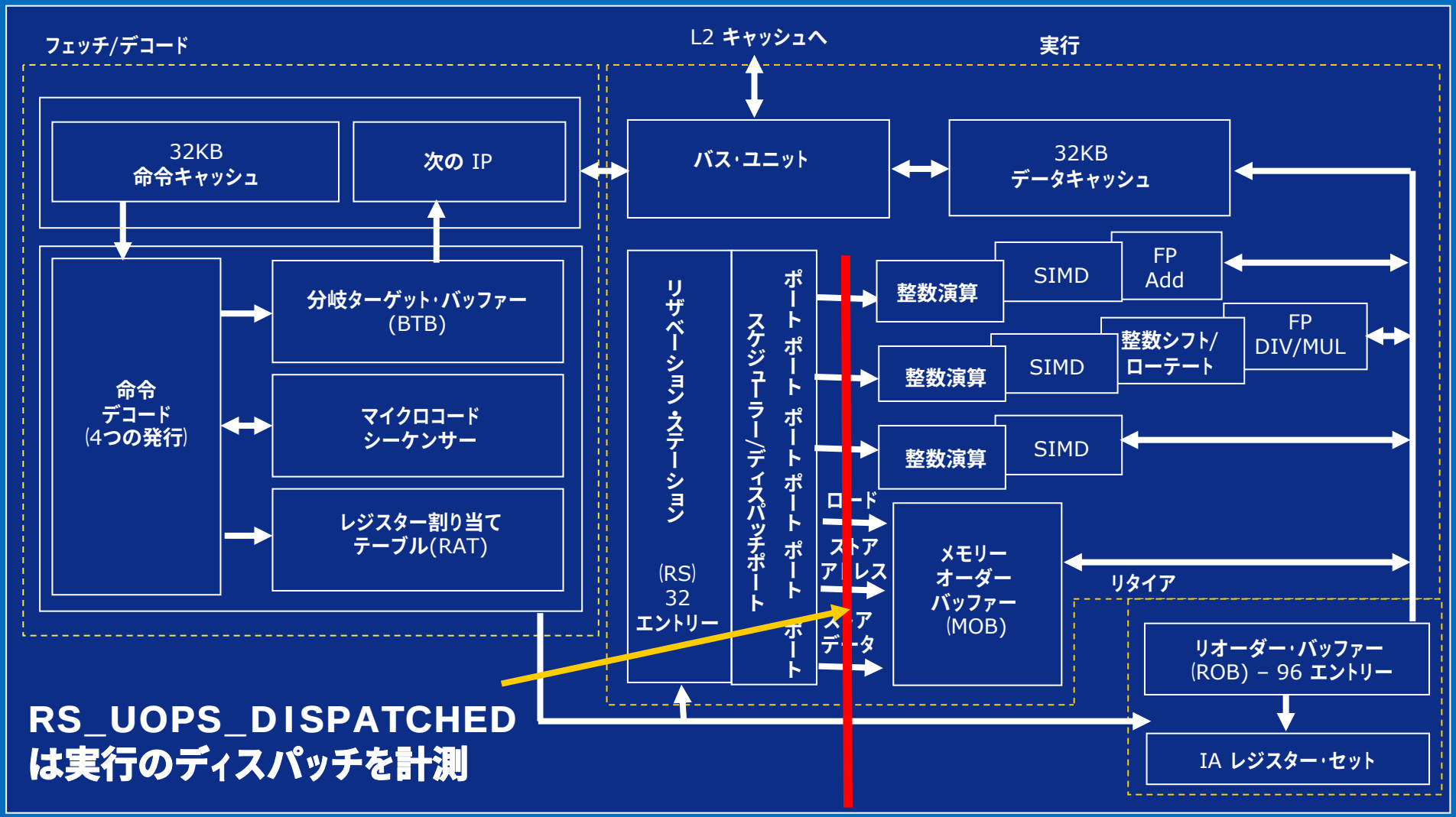
命令カウントを減らすため最適化を向上させ、命令レベルの並列性 (ILP) を増すためループを分割する

RESOURCE_STALLS.BR_MISS_CLEAR イベントでパイプライン・フラッシュによるストールを算出

UOP（マイクロ・オペレーション）の流れ

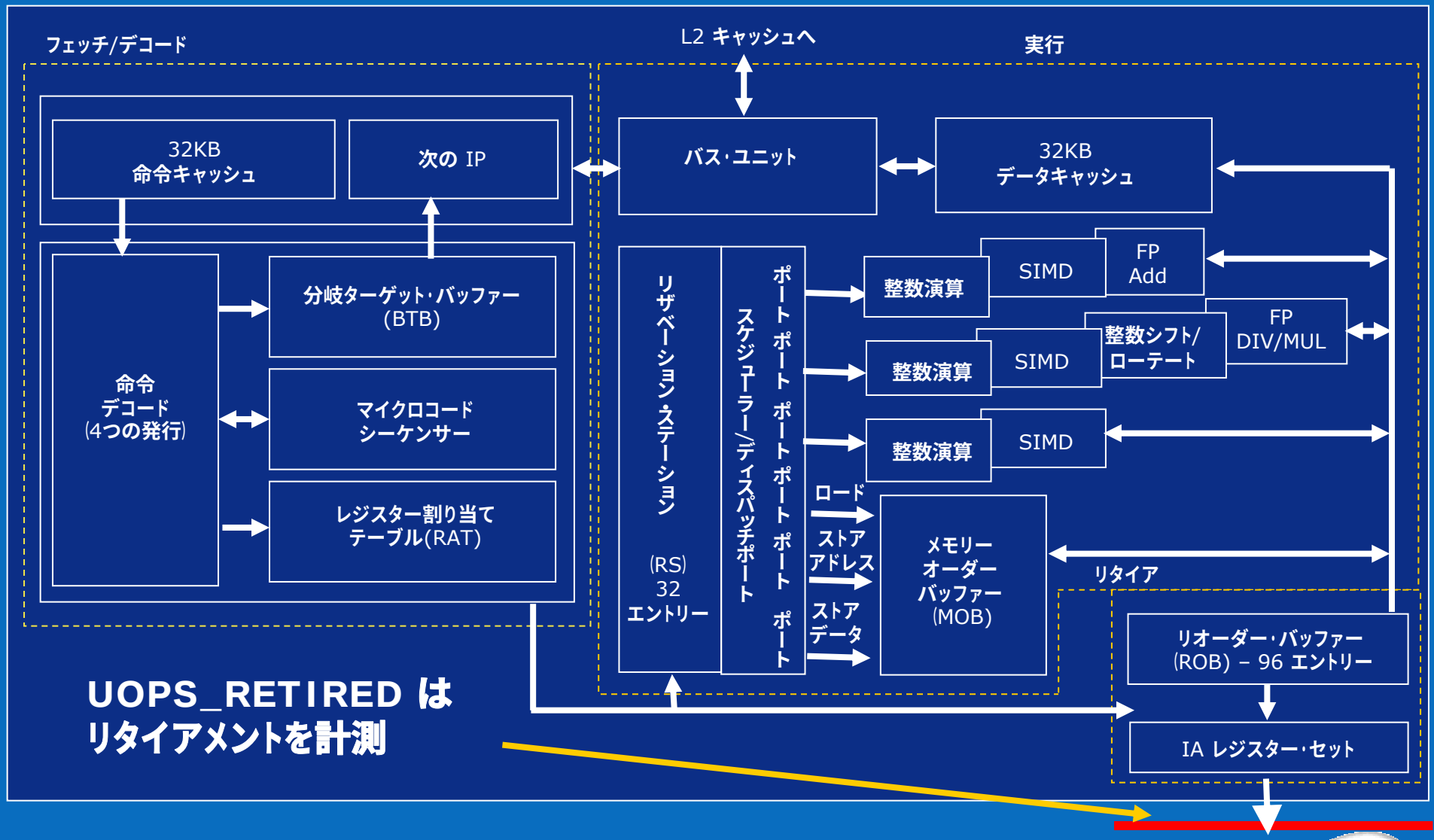


UOP（マイクロ・オペレーション）の流れ



インテル® ソフトウェア開発製品

UOP（マイクロ・オペレーション）の流れ



インテル® ソフトウェア開発製品

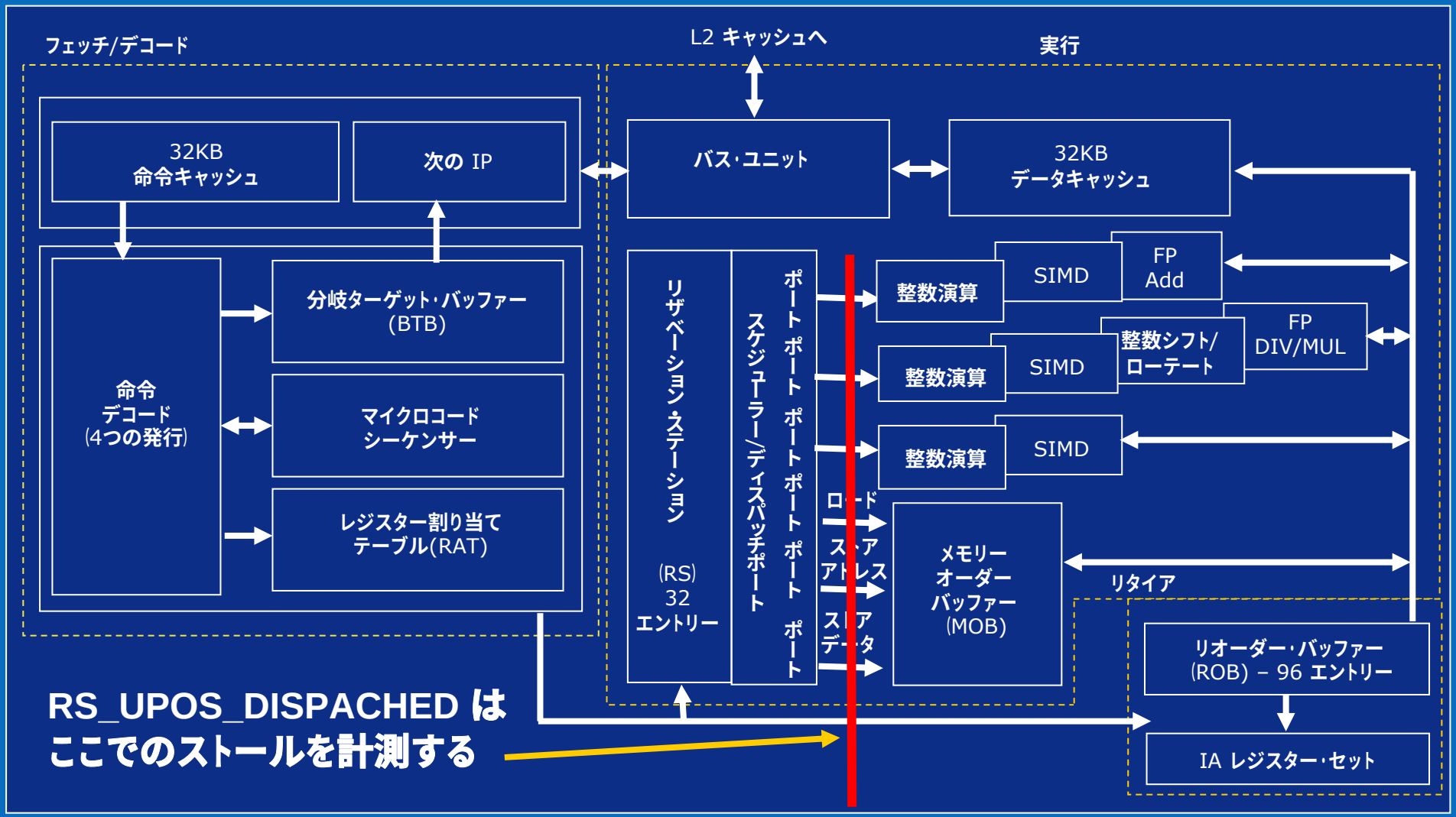
実行ステージでの有効性を計測

OOO エンジンはリザベーション・ステーション (RS) から実行ユニットへの命令ディスパッチを行う
  - ディスパッチされた命令は入力ソースが利用可能になるまで待つ
- RS_UOPS_DISPATCHED は、各 CPU サイクルで RS からディスパッチされたマイクロ・オペレーションの数をカウントする

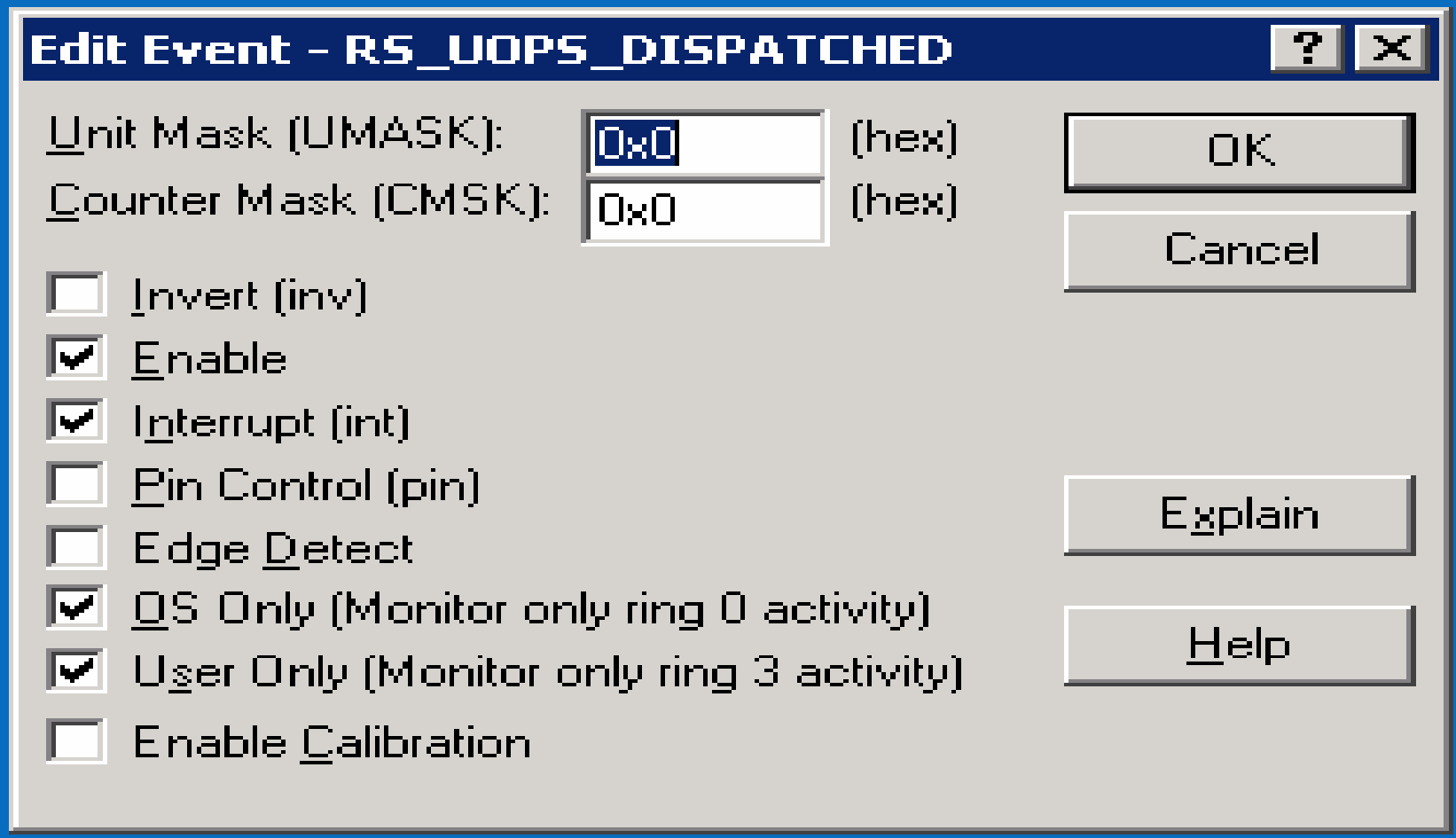
最も重要な PMU* イベント

* Performance Monitoring Unit (PMU)

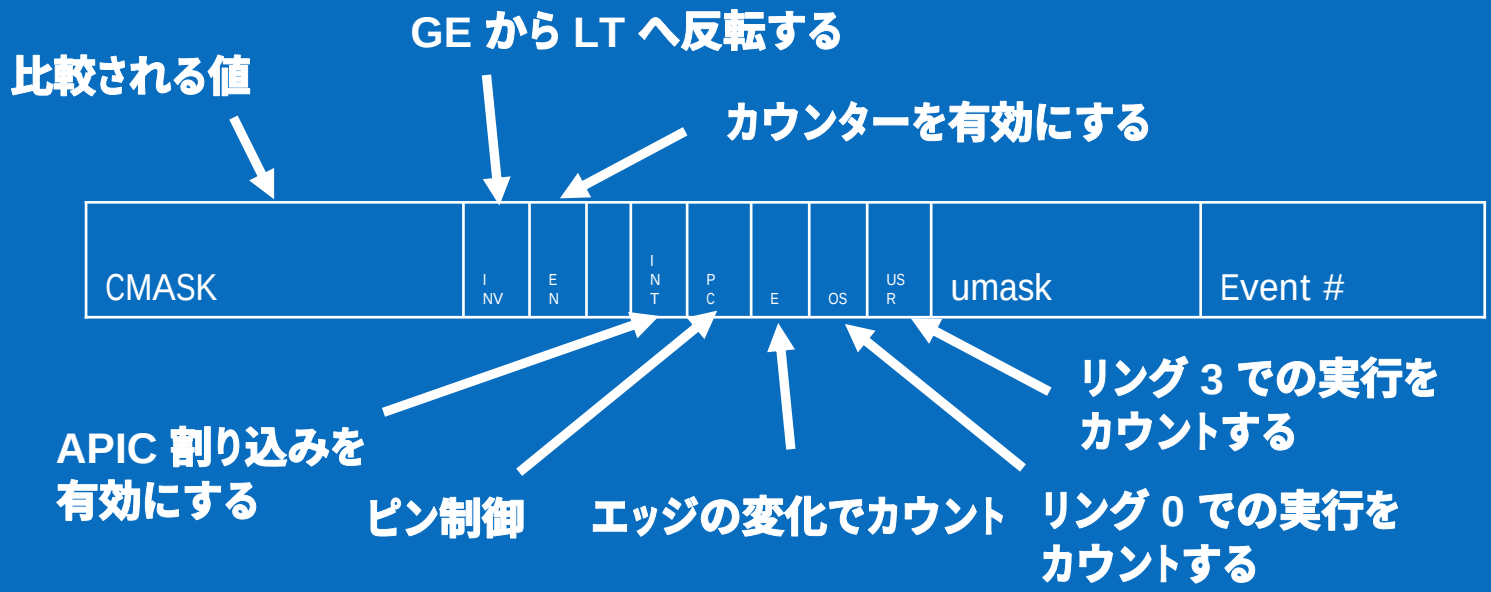
実行時に何が起こっているか



VTune™ アナライザーのイベント編集



PMU のいくつかの機能



RS_UOPS_DISPATCHED で、
CMASK = 1、INV = 1 に設定
UOPS はディスパッチされなかった == ストール
RS_UOPS_DISPATCHED.CYCLES_NONE

設定の方法論

RS_UOPS_DISPATCH: cmask = 1

トータル・サイクル ~
CPU_CLK_UNHALTED

RS_UOPS_DISPATCH: cmask = 1 : inv = 1

CPU_CLK_UNHALTED は OOO エンジンでのストールと実行を分析できる

要求 >99% CPU 利用率

もしくは ユーザー PL のみサンプリング

イベントのカウントはHALTサイクルの間も行われる

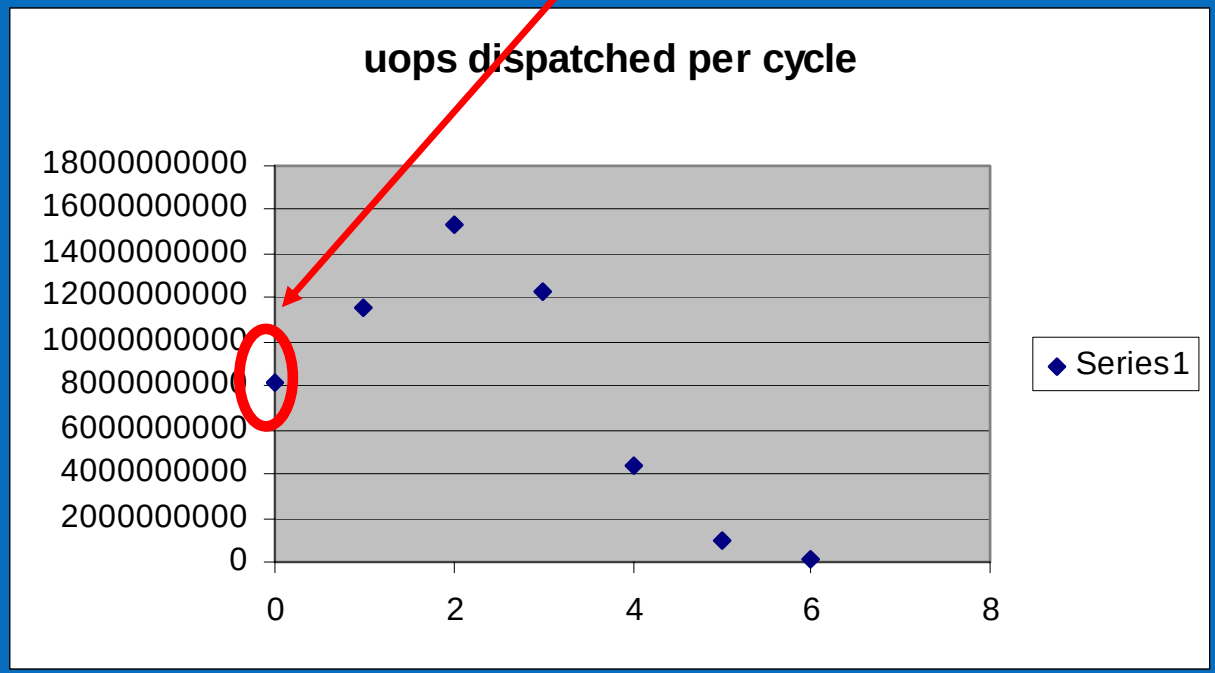
UOP / サイクルの分布

RS_UOPS_DISPATCHED:cmask=1:inv=1
RS_UOPS_DISPATCHED:cmask=2:inv=1
RS_UOPS_DISPATCHED:cmask=3:inv=1
RS_UOPS_DISPATCHED:cmask=4:inv=1
RS_UOPS_DISPATCHED:cmask=5:inv=1
RS_UOPS_DISPATCHED:cmask=6:inv=1
RS_UOPS_DISPATCHED:cmask=7:inv=1

N-1 の値を引く

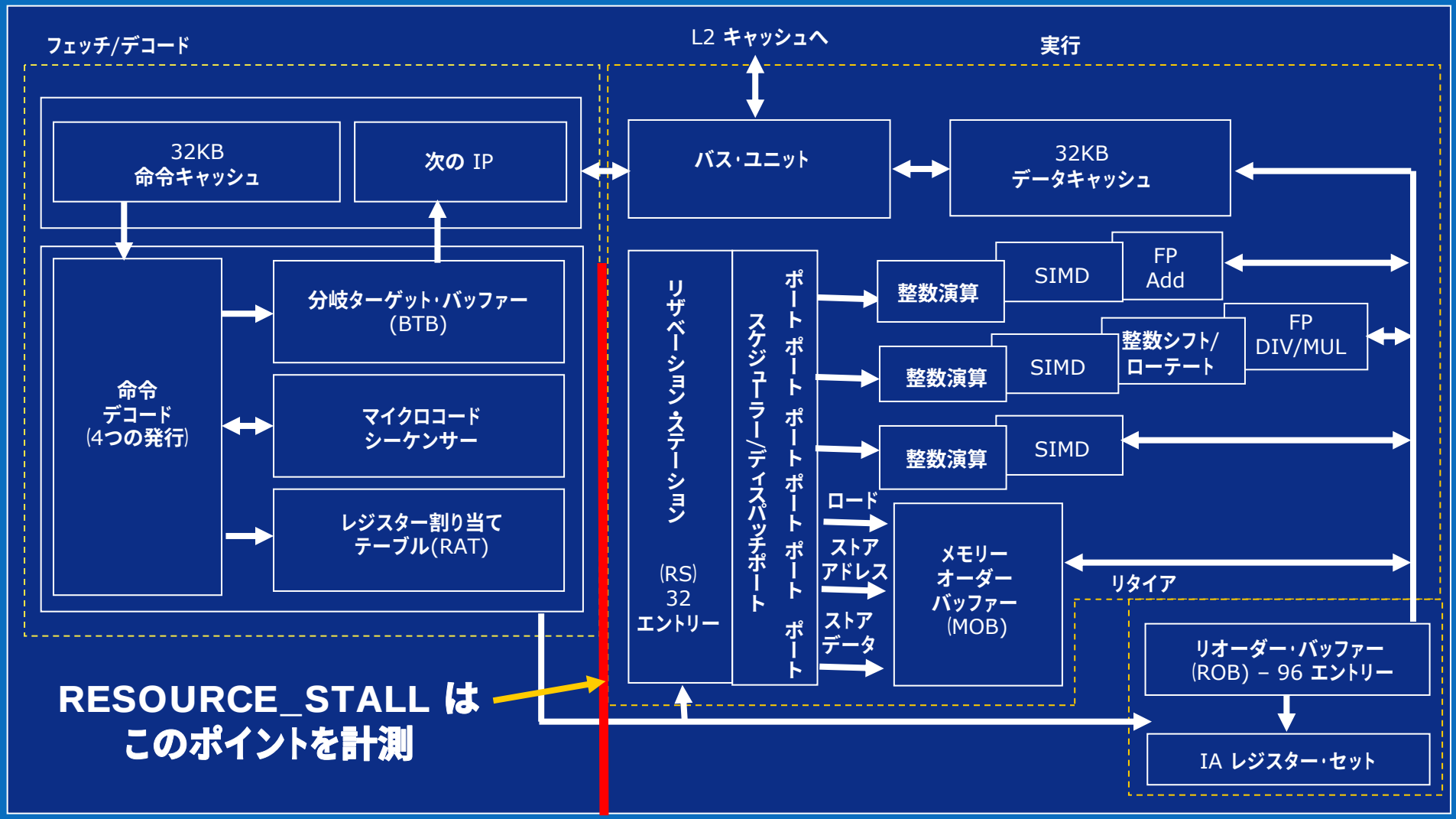
ストール サイクル

命令レベルの並列性の分布
(例: a[i] = exp(x[i]); 1つのループ中)



インテル® ソフトウェア開発製品

アーキテクチャー・ブロック・ダイアグラム

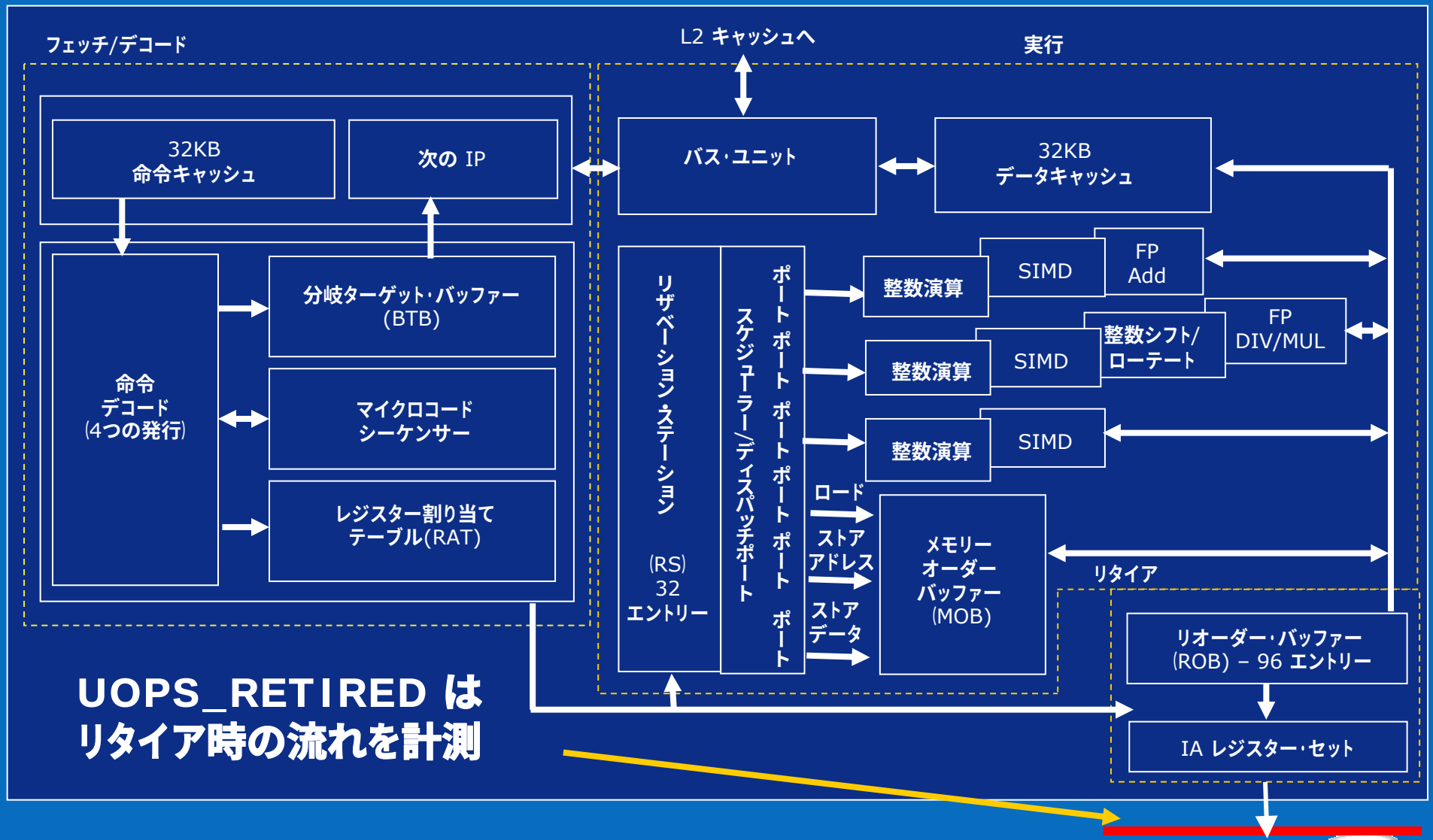


インテル® ソフトウェア開発製品

RESOURCE_STALL の要素

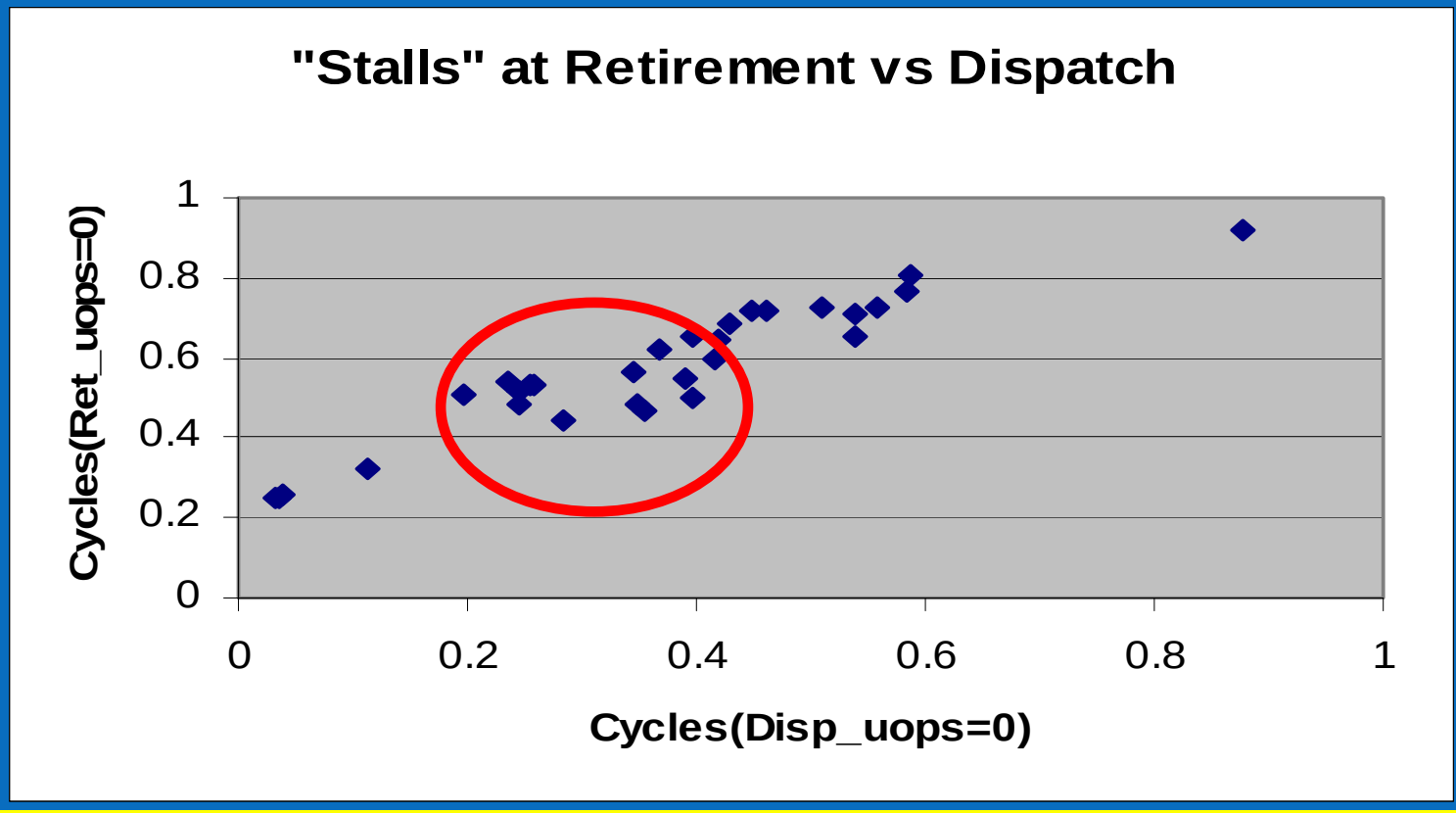
- OOO エンジンへの UOP の流れは下流の原因でブロックされる
- RESOURCE_STALLS.BR_MISS_CLEAR
  - 分岐予測ミスによるパイプライン・フラッシュによるストール
  - FP 命令に続く分岐予測ミスは、RESOURCE_STALLS.CLEAR に含まれる
- RESOURCE_STALLS.ROB_FULL
  - ROB 中に 96 個の命令がある
- RESOURCE_STALLS.LD_ST
  - すべてのストアもしくはロード・バッファーが使用中
- RESOURCE_STALLS.RS_FULL
  - RS で 32 個の命令が入力待ちの状態

アーキテクチャー・ブロック・ダイアグラム



インテル® ソフトウェア開発製品

リタイアとディスパッチによるストール、どちらを解決するか？



ループ中の差は OOO 実行による影響
ストールがディスパッチで発生している場合、
わずかな誤検出がある

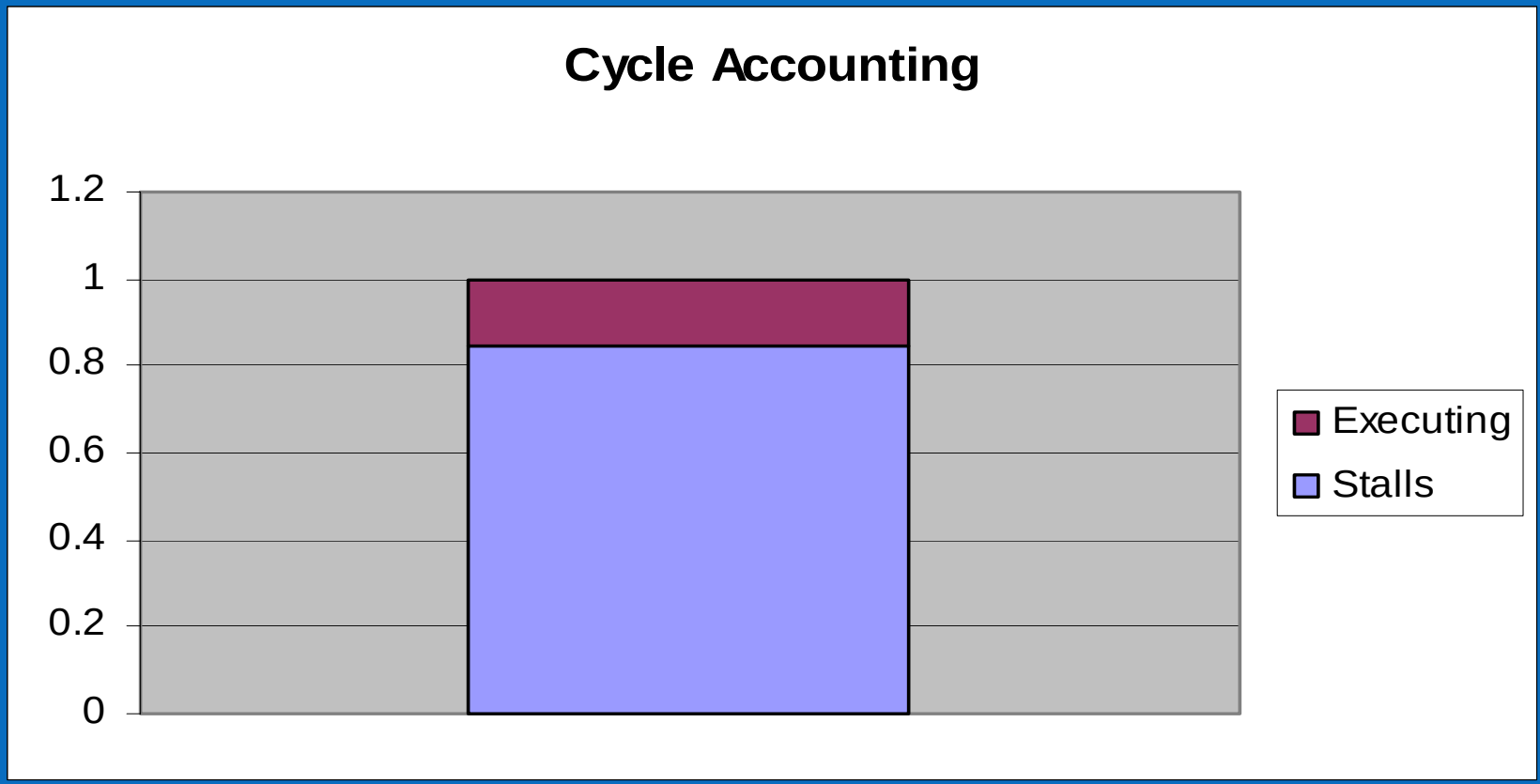
X86 におけるサイクル計算

- **サイクル** ＝
  rs_uops_dispatched.cycles_none + rs_uops_dispatched:cmask=1
  “ストール” ＋ ディスパッチ
  - **定義によって均等**
- **サイクル** ~ CPU_CLK_UNHALTED.CORE
  - CPU **依存の高いアプリケーション向け**

X86 におけるサイクル計算

- ディスパッチ ~ cycles_dispatch_retiring_uops + cycles_dispatch_non_retiring_uops
  - リタイア/非リタイア UOP のオーバーラップはないと過程
    - 最悪のシナリオ
- 非リタイア UOP = rs_uops_dispatched – (uops_retired.any + Uops_retired.fused)
  - 非リタイア UOP サイクル ~ リタイアしていない uops / avg_uops_per_cycle
- 断片的に浪費された時間 = rs_uops_dispatched / (uops_retired.any + uops_retired.fused) - 1

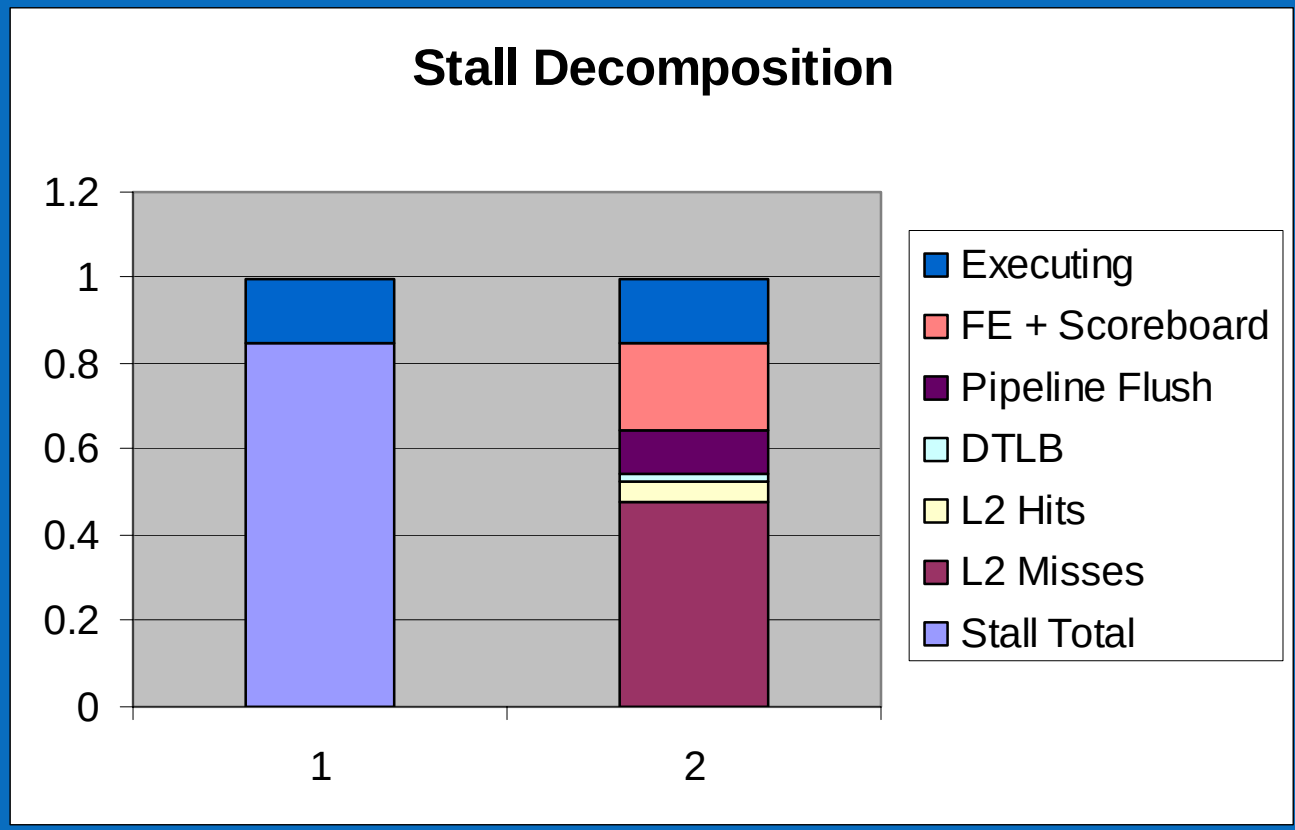
サイクル計算をまとめる



ストール・サイクル = UOP がディスパッチされていないサイクル
= RS_UOPS_DISPATCH.CYCLES_NONE

グラフは例題として作成したもので、実際の集計データではありません。

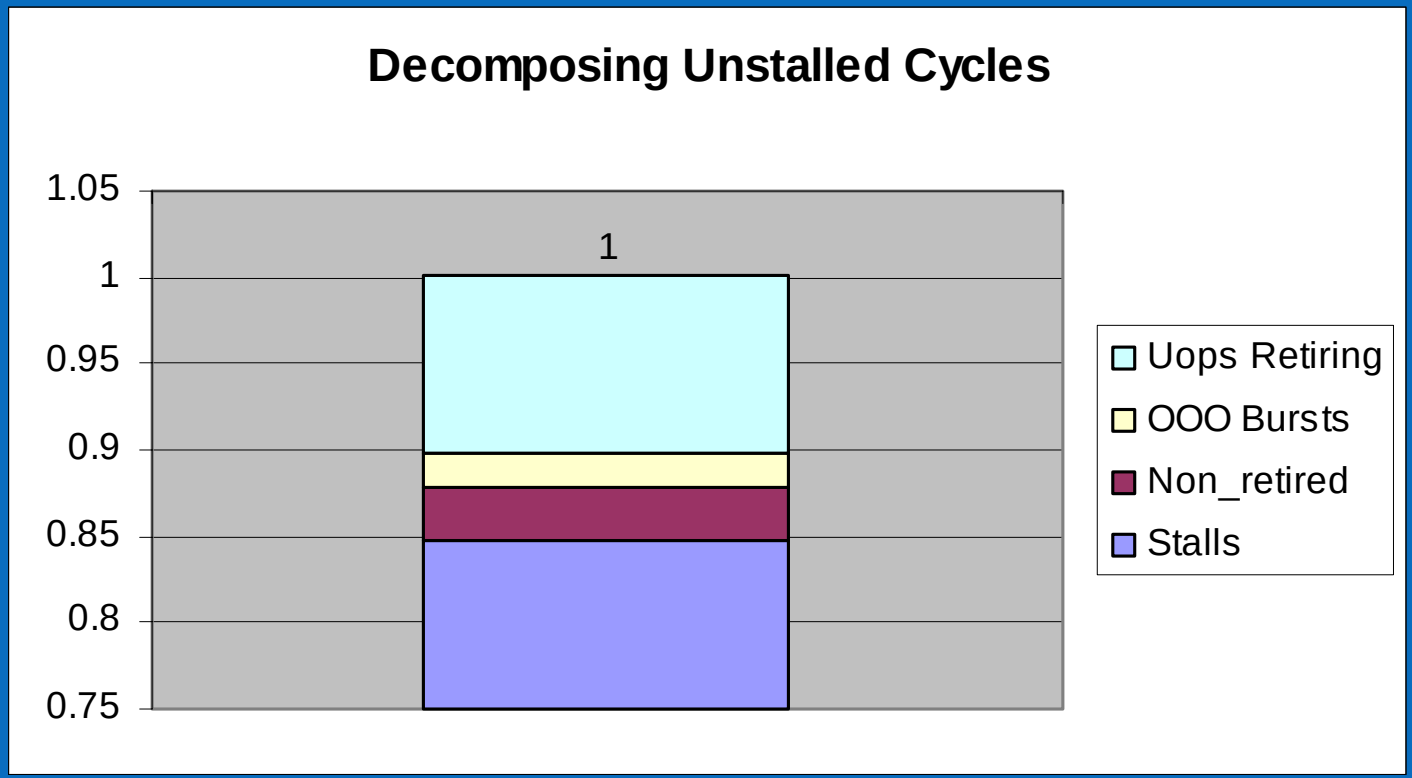
ストール・サイクルの分類



パイプライン・フラッシュ = RESOURCE_STALLS.BR_MISS_CLEAR / サイクル
L2 ヒット = (MEM_LOAD_RETIRED.L1D_LINE_MISS - MEM_LOAD_RETIRED.L2_LINE_MISS)*12 / サイクル
DTLB / L2 ミス = イベント・カウント * ペナルティー / サイクル
FE + スコアボード = ストール – 上記の全て

グラフは例題として作成したもので、実際の集計データではありません。

ストールしていないサイクルを分類

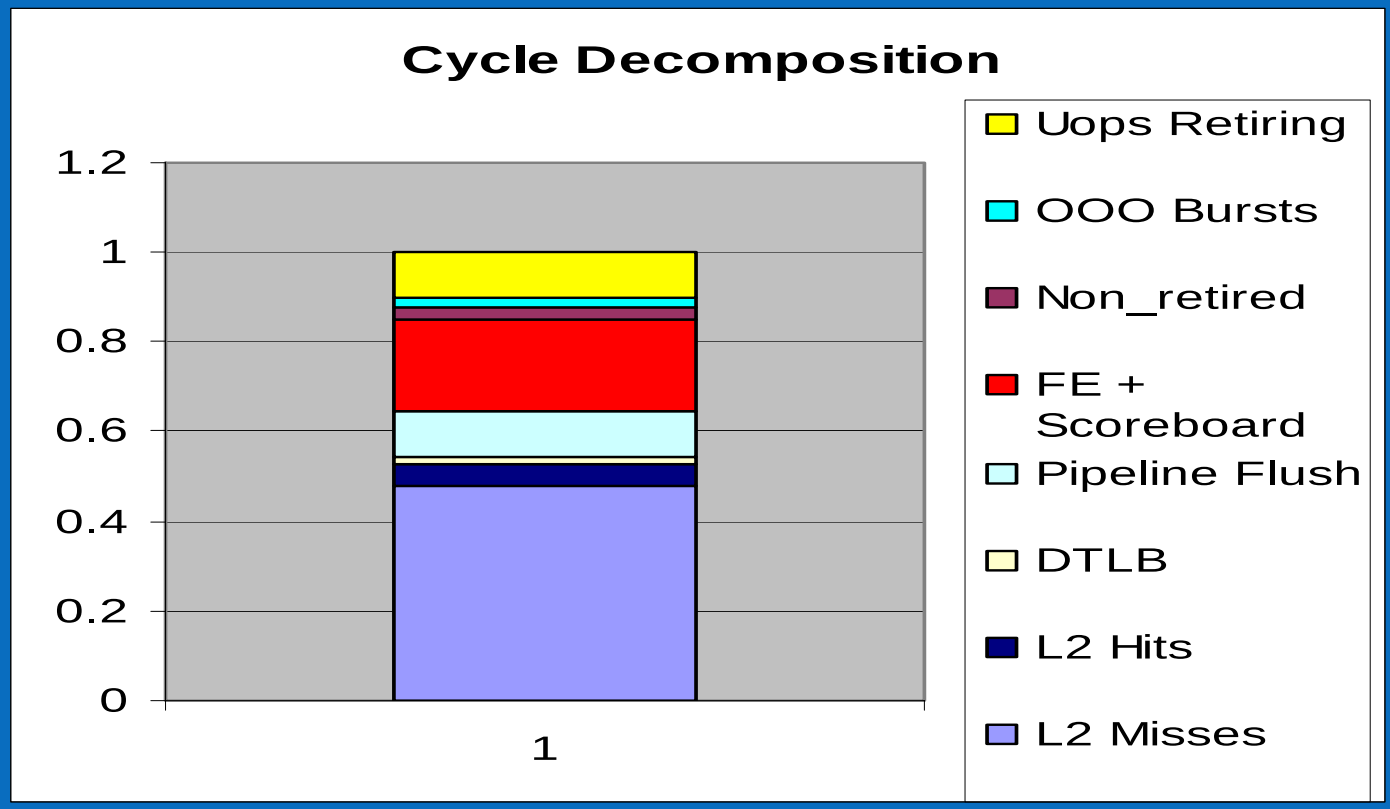


非リタイア =
((1 – (Uops_retired.any+Uops_retired.fused)/RS_Uops_Dispatched) * RS_Uops_Dispatched:cmask=1 / CPU_CLK_UNHALTED.CORE

OOO バースト = Uops_Retired.Any.cycles_none – ストール – 非リタイア

グラフは例題として作成したもので、実際の集計データではありません。

全てのサイクルを分類



オーバー・カウントのリスク / FE の最少化 + スコアボード

しかし、非効率な命令実行解決へのガイドとなる

グラフは例題として作成したもので、実際の集計データではありません。

アーキテクチャー上の問題:

問題	カウンター	ペナルティー
先行する不明なアドレスからのロード位置へのストア	Load_Blocks.ADR	~5
8 バイトの中間から 4 バイトをストアフォワードする	Load_Blocks.Overlap_Store	~6
先行する N*4096 のオフセットをもつ不明なアドレスからのロード位置へのストア	Load_Blocks. Overlap_Store	~6
L2 からの 2 キャッシュ・ラインのロード	Load_Blocks.UNTIL_RETIRE	~22
先行するストアを伴う L2 からの 2 キャッシュ・ラインのロード	Load_Blocks.UNTIL_RETIRE	~20
プリフィックスによる読み込みレングスの変更 (16 ビット即値)	ILD_STALLS	~6

“FE + スコアボード” へ影響する

4つの大きな変更

CPU_CYCLES ⇒ CPU_CLK_UNHALTED.CORE

BACK_END_BUBBLE.ALL ⇒ RS_UOPS_DISPATCHED.CYCLES_NONE

BUS_MEMORY.ALL.SELF ⇒ BUS_TRANS_ANY.SELF

DEAR_LATENCY_GT_64 ⇒ MEM_LOAD_RETIRED.L2_LINE_MISS

サイクル、ストール、プリフェッチされていないロードおよびバンド幅

より詳しいイベントの階層

第 1 レベルのイベント	SAV (Sample After Value)
CPU_CLK_UNHALTED.CORE	2,000,000
RS_UOPS_DISPATCHED.CYCLES_NONE	2,000,000
UOPS_RETIRED.ANY + UOPS_RETIRED.FUSED	2,000,000
RS_UOPS_DISPATCHED	2,000,000
MEM_LOAD_RETIRED.L2_LINE_MISS	10,000
INST_RETIRED.ANY_P	2,000,000
ループ	
BUS_TRANS_ANY.SELF	100,000
BUS_TRANS_ANY.ALL_AGENTS	100,000
分岐	
RESOURCE_STALLS.BR_MISS_CLEAR	2,000,000

SAV 値はサンプルの比率となり、ペナルティーを吸収する

イベントの階層

第 2 レベルのイベント	SAV (Sample After Value)
MEM_LOAD_RETIRED.DTLB_MISS	20,000
MEM_LOAD_RETIRED.L2_MISS	10,000
MEM_LOAD_RETIRED.L1_LINE_MISS	200,000
BR_CND_MISSP_EXEC	2,000,000
BR_CND_EXEC	2,000,000
BR_CALL_EXEC	200,000
BR_CALL_MISSP_EXEC	200,000
ILD_STALLS	200,000
LOAD_BLOCK.STORE_OVERLAP	200,000

SAV 値はサンプルの比率となり、ペナルティーを吸収する

例: L1 ミス / L2_hit のペナルティーは 10 サイクル

ループにおける方法論？

- ホットな関数を探し最適化を行う
  - アライメントの解決、ベクトル化を促進するためループを分割
- バスのバンド幅に制限される関数を特定
  - FP 命令に制限されるループとマージする
- L2 キャッシュ・ミスを特定しソフトウェア・プリフェッチを行う
- OOO エンジンの流れを最適化する
  - ループの分割が有効

まとめ

インテル® Core™ 2 プロセッサーはループを効率よく実行する

- 2 つの SIMD 命令を 1 サイクルで実行
- みじかなパイプライン

インテル® Core™ 2 プロセッサーの PMU は優れている

- バンド幅を簡単にそして明確に定義可能
  - アドレス・バスの利用率は無いと仮定
- より多くのプリサイス・イベント
  - 命令リタイアとそのコンポーネント
  - LLC ミス
- ループ中の非効率な命令実行は予測可能

参考資料

リタイアイベントと非リタイアイベント

リタイアしたイベントとは、マシンステートを確定した命令によるイベントのみを含む

- 例：MEM_LOAD_RETIRED.L2_MISS（L2キャッシュのミスで発生したロードの数）

非リタイアイベントは、マイクロアーキテクチャーにおけるアウトオブオーダーの推測による流れで発生する

インテル® Core™ アーキテクチャーにおけるプリサイス・イベント

BR_INST_RETIRED.MISPRED

INST_RETIRED.ANY_P

MEM_LOAD_RETIRED.DTLB_MISS

MEM_LOAD_RETIRED.L1D_LINE_MISS

MEM_LOAD_RETIRED.L1D_MISS

MEM_LOAD_RETIRED.L2_LINE_MISS

MEM_LOAD_RETIRED.L2_MISS

SIMD_INST_RETIRED.ANY

X87_OPS_RETIRED.ANY

これらのイベントはプリサイス・イベントと呼ばれ、イベントが発生した時の命令実行ポインター（EIP）に即座に関連付けられる。そのため、イベントの設定にイベント・スキッドは無い。

CPU_CLK_UNHALTED.CORE および INST_RETIRED.ANY イベントで開始

CPU_CLK_UNHALTED.CORE (CPUがHALT中では無いコアサイクル) イベントで CPU サイクルを測定

CPU_CLK_UNHALTED.CORE /プロセッサーの周波数 = 1秒あたりの回数

INST_RETIRED.ANY (リタイアした命令の数) =プロセッサー・ステートが確定する (完全に実行される) 命令の数

CPI (命令あたりのサイクル数) = CPU_CLK_UNHALTED.CORE / INST_RETIRED.ANY

CPI の値が高い場合、最適化の余地がある

CPU_CLK_UNHALTED.CORE と CPU_CLK_UNHALTED.NO_OTHER

CPU_CLK_UNHALTED.NO_OTHER (コアがアクティブでもう一方のコアが HALT 中のバスサイクル) イベントをバスの周波数で割ることで、1つのコアがどれくらいバスを占有しているかが解る

CPU_CLK_UNHALTED.CORE イベントは、プロセッサー HALT 状態ではないサイクルを測定する

CPU_CLK_UNHALTED.NO_OTHER を監視することで、アプリケーションやシステムが利用するバスの並列利用度が解る

まとめ

インテル® VTune™ パフォーマンス・アナライザーを使用すると、ソフトウェアに含まれているマイクロアーキテクチャー上のボトルネックを検出できる

インテル® Xeon® プロセッサーおよびインテル® Pentium® 4 プロセッサー上でのソフトウェアの最適化に関する資料は、インテルの Web サイトを参照

インテルでは、以下のような情報を提供

- 日本語プロセッサー・テクニカル・ドキュメント
  - http://www.intel.co.jp/jp/developer/download/
- インテル® ソフトウェア開発製品
  - http://www.intel.co.jp/jp/developer/software/products/
- テクニカル情報、サポートしている環境、ホワイトペーパー
  - http://developer.intel.com (英語)
- 製品サポート
  - http://support.intel.com (英語)
- フォーラム
  - http://softwareforums.intel.com/ids (英語)

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。
製品に付属の売買契約書『Intel's Terms and conditions of Sales』に規定されている場合を除き、インテルはいかなる責を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的所有権を侵害していないことへの保証を含む) に関しても一切責任を負わないものとします。

インテル製品は、予告なく仕様が変更されることがあります。

* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。