

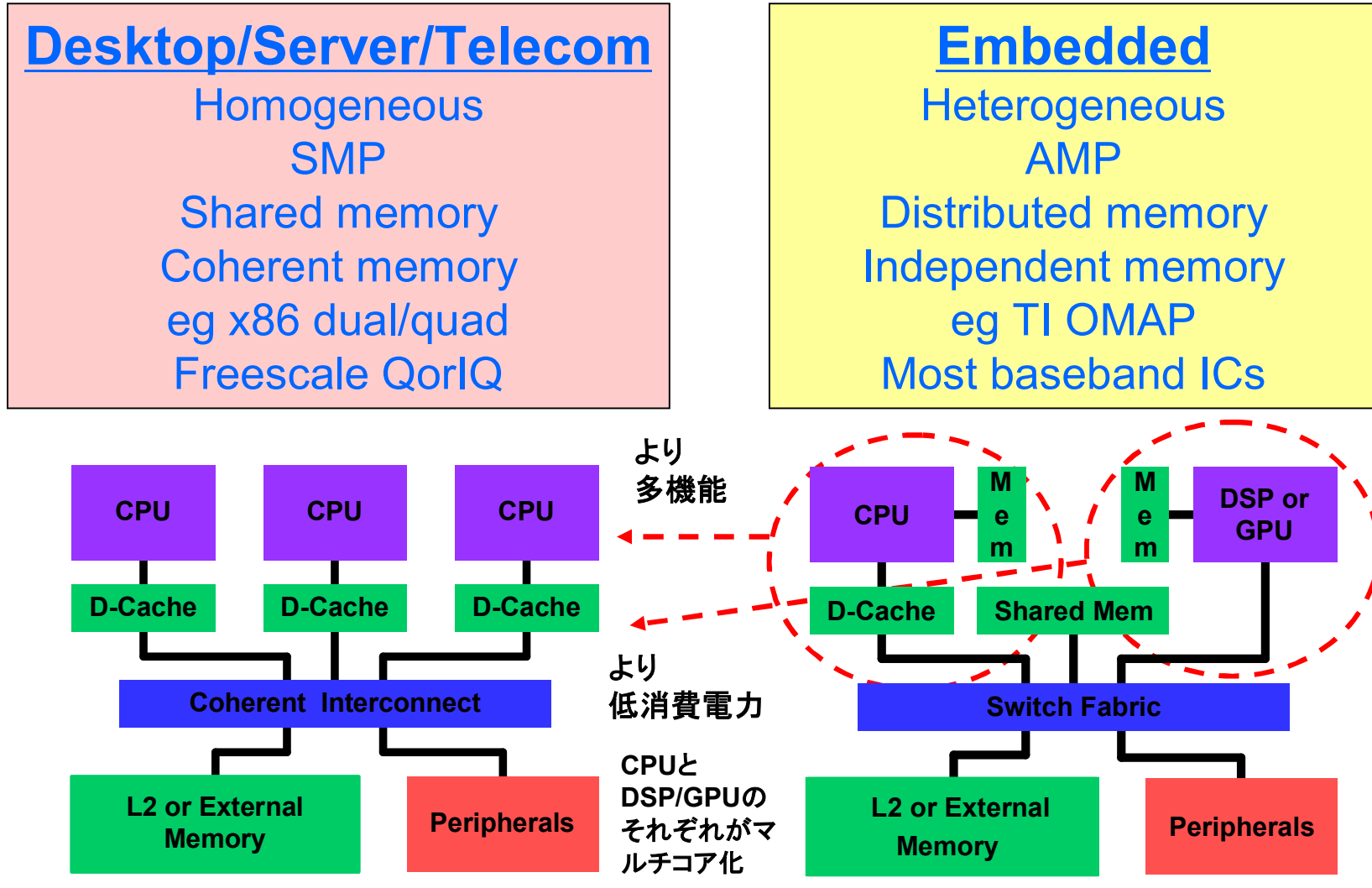


ハードウェアスレッドと
マルチコアを
最大限に活用する開発環境



- ・ マルチコア化の問題点と検討課題
- ・ Criticalblueのソリューションの概略
- ・ デモ

マルチコアは既存の技術だが？

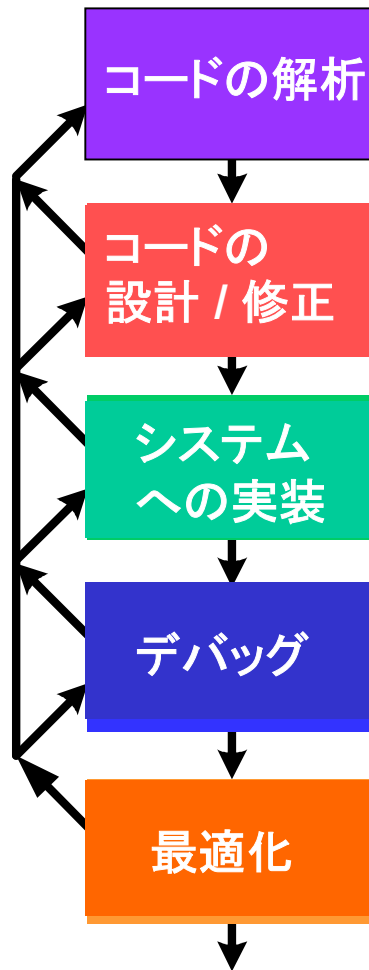


マルチコアのソフト開発: 何が難しい?

- VDC レポート: マルチコアを含んだ製品を開発する際の
挑戦/ギャップとの遭遇のトップ 5
 - ソフトウェア技術者へのアンケート結果

ソフトウェアの開発ツール	55.4%
ソフトウェアのプログラミングモデル	50.0%
ソフトウェアの性能解析	39.1%
ソフトウェアのドライバー/ミドルウェア	22.8%
ダウンロード/接続の速度	12.0%

既存のコードのマルチコア化の現状



- 最適なタスク分割を検討し、コードをスレッド化
 - 粒度が大きすぎると並列実行の効果が減り、小さいとタスク切り替えのオーバーヘッドが増大
- シミュレーションや開発ボードで検証
 - コード修正に伴い正常動作させるまでも大変
 - 期待した性能に未達だと原因究明は困難
 - タスク間のデータ依存による待ち時間か、タスク切り替えのオーバーヘッドか？
- 原因を究明し、コードを変更して再実行するという繰り返しループに時間がかかる
 - しかも達成可能な性能値の目標設定が困難

今マルチコアに移行すべきか？

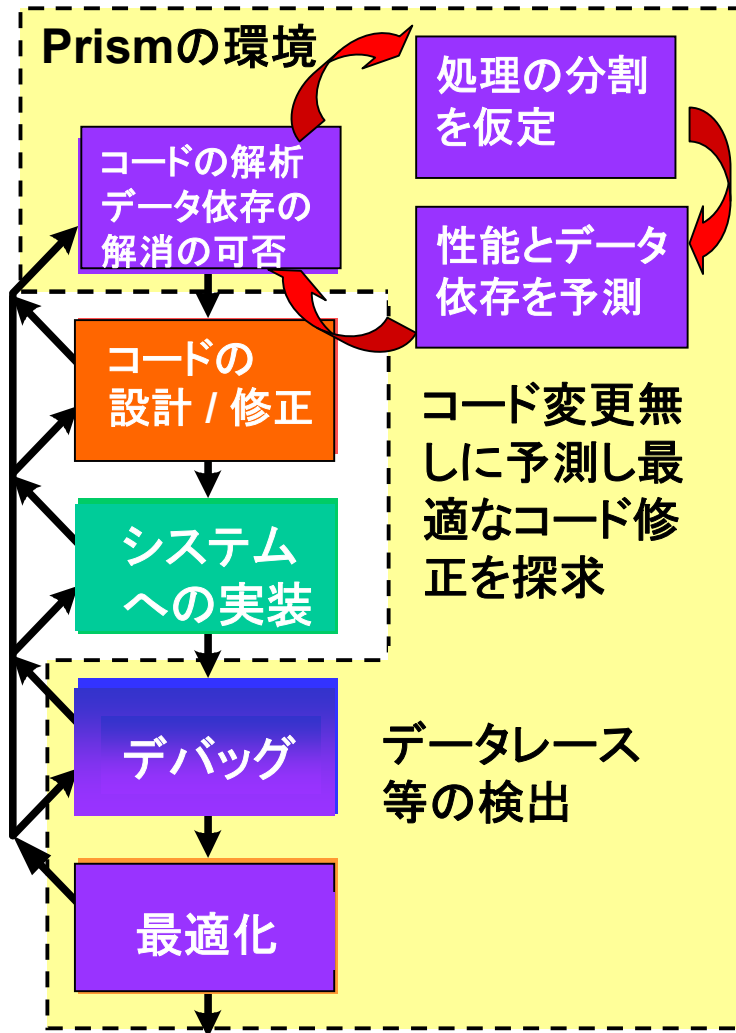
- ・ シングルコアで済ませられないか？
 - ハードウェアスレッドやキャッシュの最適化では無理か？
 - 既存コードの最適化で済ませられないか？
- ・ 対象のアプリケーションはマルチコア向きか？
 - コードはどの位の変更が必要か？
 - 性能はどの位上がるか？

Criticalblueのご紹介



- ・ 会社概要
 - 2003年にUKスコットランドのエジンバラで設立
 - プロセッサのアーキテクチャ設計やEDAと組み込みソフトウェア分野の専門家の集団
 - 組み込みソフトウェアの性能向上をヘルプする製品群を提供
- ・ Multicore Associationへの貢献
 - マルチコア・プログラミング・作法 (MPP)策定分科会 の Co-chairにCEOのDavid Stewart が就任
 - MPPの作成作業を分担
 - マルチコア用の並列化プログラムの浸透をツールサイドからサポート

CriticalBlue 社のPrism の概要



Prismとは

- 従来のソフトウェアの開発環境に付加して、システムの最適化や、マルチコア化を手助けする専用ツール
- 前述のプログラムの処理の分割とデータ依存の解析を、**コードの修正と実行無しに仮想的に行う事が可能**
- Prism を付加すると並列化のフローは、左図の様に改善

Prismを使用した設計フロー

- 現在の設計手法の変更は不要
 - 既存の実行ログとバイナリが入力
- 従来の一連の繰り返し作業を、コード修正と再実行無しに、仮想環境で実現
 - もしキャッシュの構成を変えたら？ハードウェアスレッドを使用したら？
 - もしこの関数をスレッド化したら性能は？その場合どこにデータの依存関係が発生する？データの依存関係が解消できたら性能は？
 - もしコアの数を増やしてマルチコア化したら性能は？
- ターゲット性能に達したら、実際に関数のスレッド化と、必要なデータの依存関係の解消の為にコード修正を行う
 - 最後に修正結果をこの環境で確認

キャッシュやハードウェアスレッドの解析

- 最適なキャッシュ構成の検討と、ハードウェアスレッドの効果の解析

Cache Performance

Global task/function filtering is enabled. Not all tasks/functions may be shown in this view.

Schedule Tile Tile ID# 0 Architecture Analysis [2 Local Caches, 1 Global Cache]

Location (Task, Function, Line) Stalls (Cyc... Miss Co...

Total	279900	9768
util_loadWithBitmap	118520	3650
main	94640	3082
util_writeImageToBitma	50080	1909
applyFilters	8920	666
cb_image_difference	7640	461

Miss Count Distribution

Task Cycles Distribution (util_loadWithBitmap)

Local L1 Local L2 Shared LLC Active Pipeline Stall Data Cache Stall

ハードウェアスレッドの数の指定

キャッシュの構成の指定可能

Core Configuration

Core Name	Logical Core Count	Enabled
Core 0	2	<input checked="" type="checkbox"/>

Cache Configuration

Enable cache modelling

Scope / Level	Line Length (bytes)	Cache Size (byt...	Access Latency (Cy...	Enabled
Local L1	32	1024	0	<input checked="" type="checkbox"/>
Local L2	32	2048	0	<input checked="" type="checkbox"/>
Local L3	32	32	0	<input type="checkbox"/>
Local L4	32			<input type="checkbox"/>
Shared LLC	32			<input checked="" type="checkbox"/>

Memory Access Latency: 40 Cycles

外部メモリとの1ライン分のアクセスサイクル数を指定

パイプラインストールの解析

- パイプラインストールの原因を解明し、システムの性能を向上

実行サイクル、分岐予測ミス、インターロックストールの解析

```

1,080 // Output
7,920 for (x=0; x<aXlimit; x++)
7,920     for (y=0; y<aYlimit; y++)
7,920         writeGreyScalePixel(pImage, row + x, col + y, aPixel);
    }
    return IMG_OK;
5 }
    
```

項目毎にソート可能

関数毎、行毎、命令毎に表示を切り替え可能

Location	Execution Count	Branch Misprediction Count	Branch Misprediction Stall T...	Interlock Stall Time
pixellate.c, 38	7,920	0	0	0
pixellate.c, 37	7,920	1,080	5,400	0
pixellate.c, 47	7,920	0	0	23,760
pixellate.c, 46	7,920	0	5,400	0
utils.c, 105	7,920	0	0	0
utils.c, 100	7,920	0	0	0
utils.c, 103	7,920	0	0	0
utils.c, 98	7,920	0	0	0
difference.c, 30	6,336	0	0	0
difference.c, 31	6,336	0	0	0
difference.c, 33	6,336	0	0	0
difference.c, 36	6,336	0	0	0
difference.c, 28	6,336	0	0	0

Location	Execution Count	Branch Misprediction Count	Branch Misprediction Stall T...	Interlock Stall Time
cb_image_pixellate (pixellate...	7,920	1,080	12,645	25,125
populateImageFromBitmap (u...	7,920	1,080	925	39,600
cb_image_difference (differen...	6,336	740	3,360	19,008
cb_image_threshold (threshol...	6,336	524	3,360	19,008
util_writeImageToBitmapFile ...	6,336	144	740	19,008
flipBytes (bitmap.c, 16)	144	59	295	0
fread	85	0	0	0
fwrite	68	0	0	0
free	40	0	0	0
malloc	34	0	0	0

対応するソース行を表示

既存のコードを解析し、タスク分割とコアへのマッピングを指定

- 関数プロファイルから適度な粒度でタスク分割を指定
- コアの数や専用と汎用を指定
- 各コアに処理するタスクをアサイン(SMPかAMPか混在)
 - Prismがマッピングエラーをチェック

Global task/function filtering is enabled. Not all tasks/functions may be shown in this view.

Function Name	Type	Mapping	Call...	Self ...	Total Cy...	Total Cy...
cb_image_pixellate	f	Core 0, Cor...	5	126,460	130,660	19.1%
_divis3	f		150	4,200	4,200	0.6%
cb_image_difference	f	Core 0, Cor...	4	115,324	115,324	16.9%
util_writeImageToBitmapFile	f	Core 0, Cor...	4	108,720	109,708	16.1%
bitmap_writeBitmapToFik	f		4	504	688	0.1%
bitmap_createBitmap	f		4	204	220	0.0%
bitmap_tree	f		4	48	72	0.0%
puts	f		4	8	8	0.0%
util_loadWithBitmap	f	Core 0, Cor...	4	80	204,804	30.0%
populateImageFromBitma	f		4	203,740	203,740	29.8%
bitmap_createBitmapFr	f		4	720	912	0.1%
bitmap_tree	f		4	48	72	0.0%
applyFilters	f	image proc...	4	76		
main	f	Core 0, Cor...	1	47		
cb_image_pixellate	f		4	171		
extractBackGround	f		1	37		
util_createImageFromBitr	f		1	50,935		
populateImageFromB	f		1	180		
bitmap_createBitmapl	f		17	34		
fread	f		2	4		
malloc	f		1	2		
fclose	f		1	2		
printf	f		1	2		

Task Name	Map to	Core 0	Core 1	image processor
applyFilters	One	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
cb_image_difference	So...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
cb_image_pixellate	All	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
main	So...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
util_loadWithBitmap	So...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
util_writeImageToBitm...	So...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

	Enabled	Dedicated	Mapping	Defined By...
Core 0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	cb_image_differ...	Trace
Core 1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	cb_image_differ...	User
image processor	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	applyFilters, cb_...	User

Hotspot Finder (10%) 100% Total Time (shows less)

タスク化を指定

コアを追加

専用コアの指定

専用コアと汎用のコアが同じタスクをアサインされているのでエラー

性能のチューニング

- タスク分割による性能予測と、発生するデータ依存箇所を表示
- データ依存の処理方法を仮定してコード変更にかかる工数に対する効果を予測
 - 構造を変えてデータ依存を無くすか、排他制御か
- 仮想的な条件変更の繰り返し:
 - タスク分割
 - データ依存関係の解消
 - アフィニティを考慮してコアへの特定スレッドのアサイン

この関数を異なるスレッドにしたらどうなる?

データ依存関係の元と使用先を表示

メモリアクセスが多い順に表示

もしデータ依存関係を無くせたら性能は
どうなる?

チューニング結果:
シングルコアより
38%高速

データの依存関係の解析

- データの依存関係が実行順序を規定し、並列実行の制約となる
- 仮定したスレッド分割によって発生するデータ依存箇所を特定

Data Dependency

```
A = 4 * C + 3;
B = A + 1;
A = 3 * C + 4;
```

Read After Write

Anti-Dependency

```
A = 4 * C + 3;
B = A + 1;
A = 3 * C + 4;
```

Write After Read

Output Dependency

```
A = 4 * C + 3;
B = A + 1;
A = 3 * C + 4;
```

Write After Write

- Read After Write 以外は、変数名の変更で解消可能(メモリは増加)

データ依存のタイプを知る事で依存関係の解消の難易度を判定

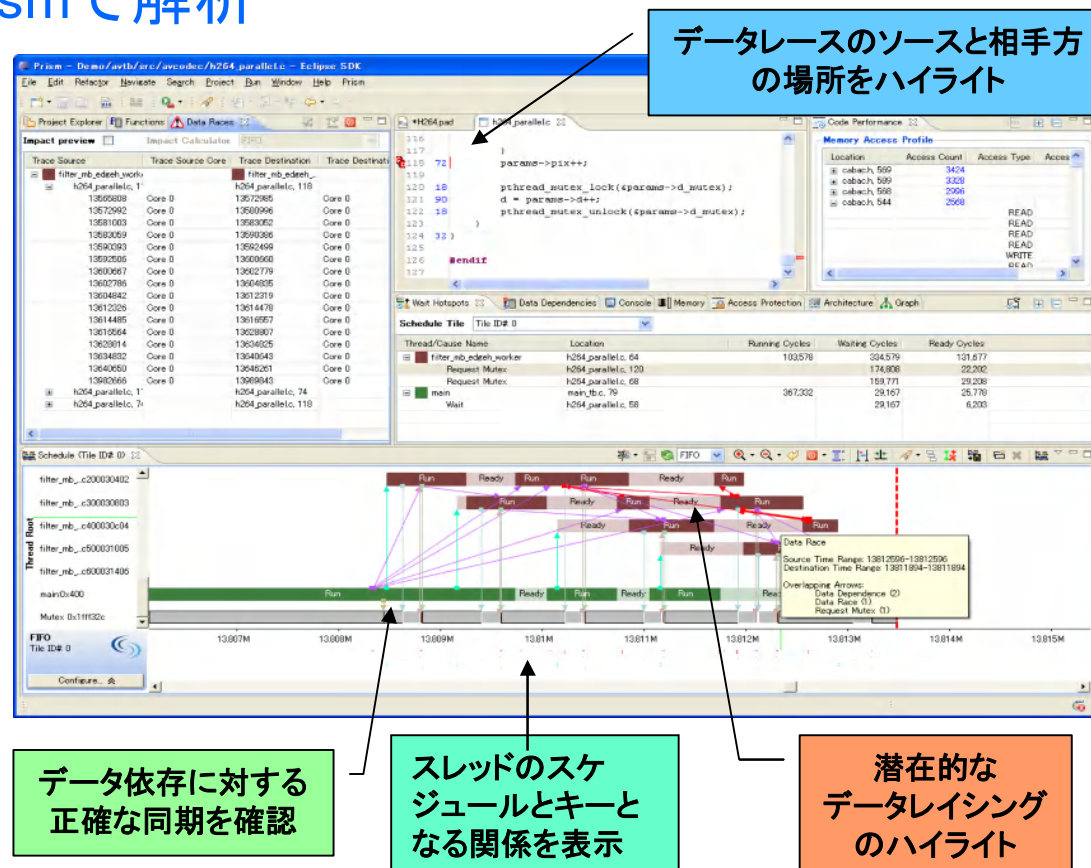
データ依存関係の元と使用先のソースを開いてその行をハイライトする指示

Source	Destination	Type
dummy.h, 208	sample.c, 83	W, R
dummy.h, 212	sample.c, 83	R, W
dummy.h, 212	sample.c, 83	R, W
dummy.h, 212	sample.c, 83	R, W
sample.c, 50	sample.c, 83	W, R
sample.c, 50	sample.c, 83	W, R
sample.c, 64	sample.c, 107	W, R

マルチコア用に修正されたコードのデバック (現状はpthread、vthreadのみサポート)

- 1コアで実行して論理バグを取る(タイミングの問題を排除)
- その実行履歴をPrismで解析
- マルチコア化を仮定した場合の問題点をPrismが提示:

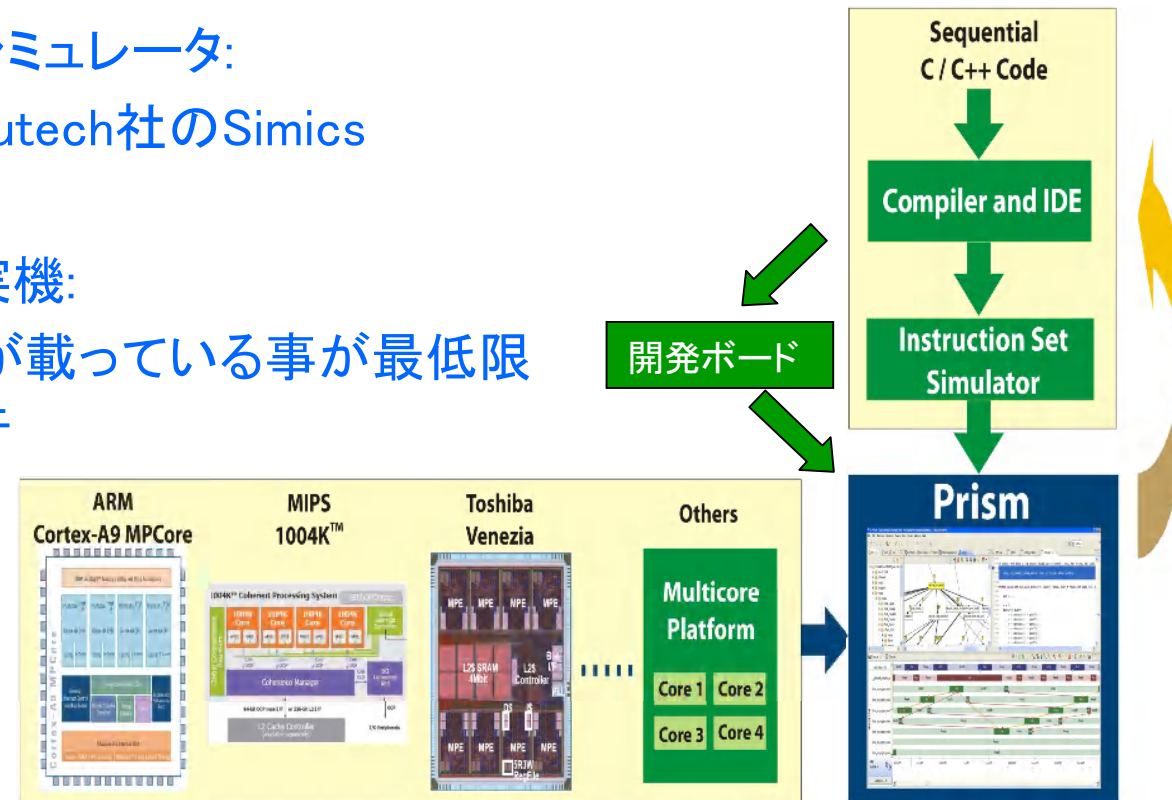
- データレースの可能性
(テストパターンでデータレースが発生していなくても可能性を指摘)



Prismと既存の環境のインターフェース

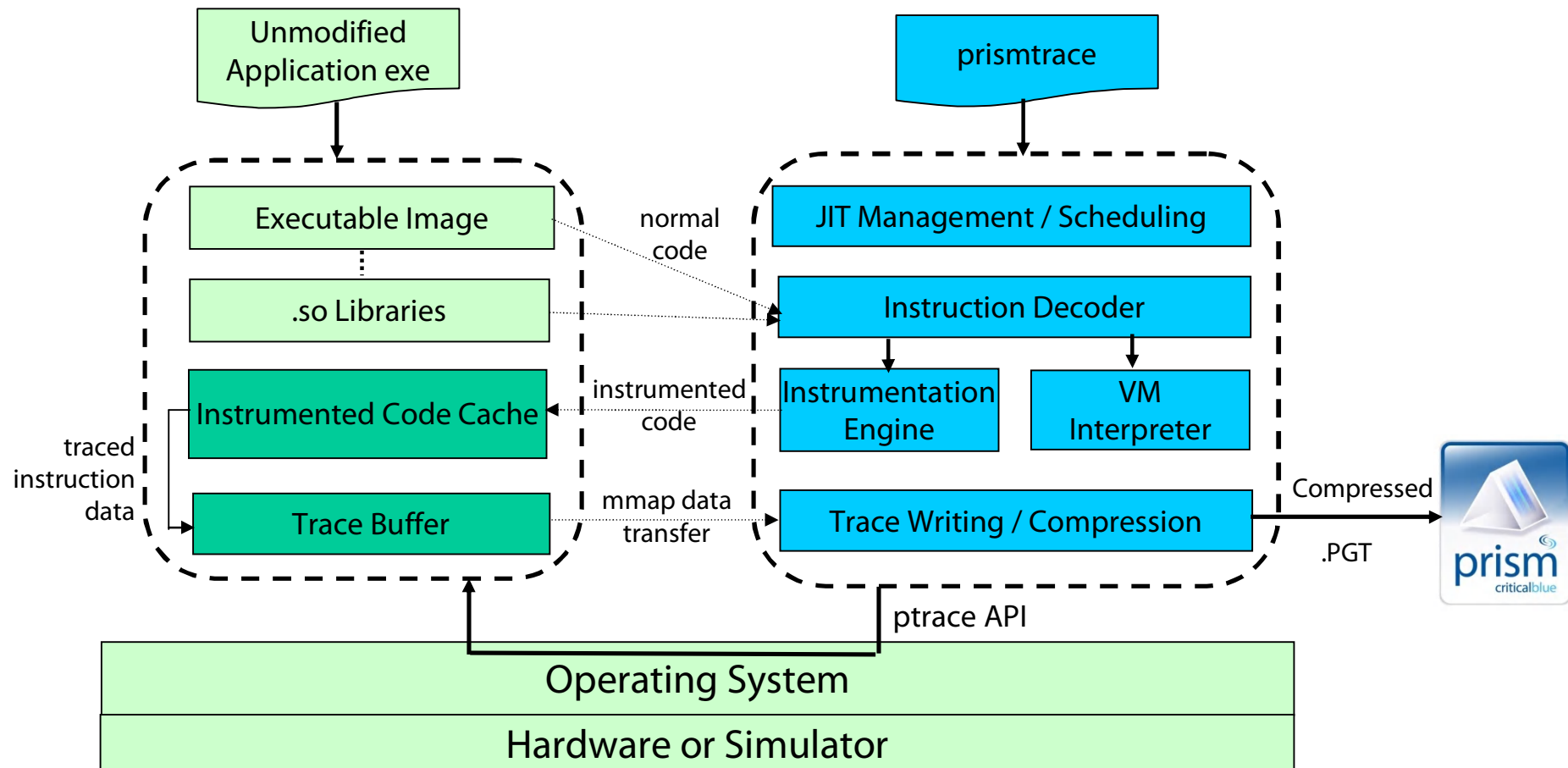
- シミュレータか実機でのターゲットソフトの実行履歴が入力
 - サポートシミュレータ:
 - 旧Virtutech社のSimics
 - QEMU
 - サポート実機:
 - Linuxが載っている事が最低限の条件

既存のソフトウェア
開発環境



共有メモリマルチコアプラットフォーム (SMPが主、AMPもサポート)

Dynamic Instrumentation の概要



次世代高速データ通信LTEの事例

- 既存のシングルコア用の LTE コードを使用
 - ダウンリンクは既に手で LTE MAC/RLC をマルチスレッド化
 - Prism を使ってアップリンクの LTE MAC/RLC をマルチスレッド化
- 最適なマルチスレッド分割:
 - 手設計: 1 週間 Prism: 2 時間
 - Force Thread と データ依存の解消などの what if' 解析を使用 (約1週間の工数削減)
- マルチスレッド化後の検証:
 - 手設計: 1-2 ヶ月 Prism: 2-3 日
 - ビジュアルなスレッド間の関係の表示が、性能解析に効果
 - Prism のデータレース箇所のハイライト機能無しには発見できなかった
 - 数人月を投入して追加のテストケースを書いたり、関連するソフトウェアのデバッグにかかる工数を削減

まとめ

- プロセスの微細化を、プロセッサコアの高性能化に割り当てる事は、低消費電力化の要求で限界化し、高性能品はマルチコア化に
 - シングルコアとマルチコアの価格差が縮小
- マルチコアの性能はソフトの並列性に依存し、製品の性能に占めるソフトウェアの重要性が増す
 - マルチコア用に最適化されたソフトウェアの開発環境(技術者の教育体制やツールの構築)の整備が製品の競争力を左右
- Criticalblueのソリューションは、従来のシーケンシャルから並列実行へのコード変更の方法を変える事無く、コード変更してのカット&トライを、コード変更せずに、仮想的にさまざまな条件を試して、最適解を探求できる環境を、既存の開発環境を変えずに提供する事で生産性を向上
 - シングルコアでの最適化と、マルチコア化の両方の検討をサポートする事で、最適なマルチコアへの移行時期を予測