


SCIMAアーキテクチャにおけるC コンパイラの実装



東京大学 田中・坂井研
中村 実

発表の流れ



⌘ 宇宙の輻射輸送計算の SCIMA 最適化

⌘ SCIMA における C コンパイラの実装

▣ newcc コンパイラ

▣ SCIMA 最適化コンパイラ

⌘ まとめ

宇宙の輻射輸送問題のSCIMA 最適化

輻射輸送問題

6次元位相空間量 (位置・方向・振動数)

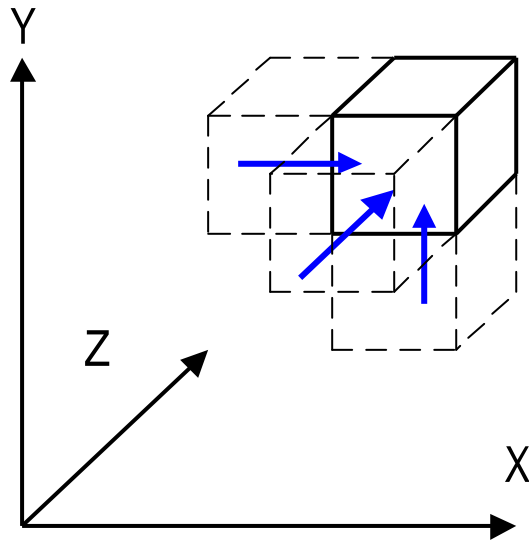
方向・振動数に関して独立 → 並列処理が可能

輻射の方向が決まった場合の計算を解く

C で記述されたオリジナルのソースコードを、手作業で最適化

輻射輸送問題

- ⌘ 3次元空間をセルに分割 (空間格子数 N^3)
- ⌘ 1つのセルは、隣接するセルのうち輻射の進行元の3つのセルを参照して値を更新
- ⌘ 空間全体のセルを計算 ($(N-1)^3$ の繰り返し)



1つのセルは 6つのプロパティを含む (6つの配列)

1つのセルの演算数はデータ依存 (exp関数が含まれる)

輻射輸送問題の性質

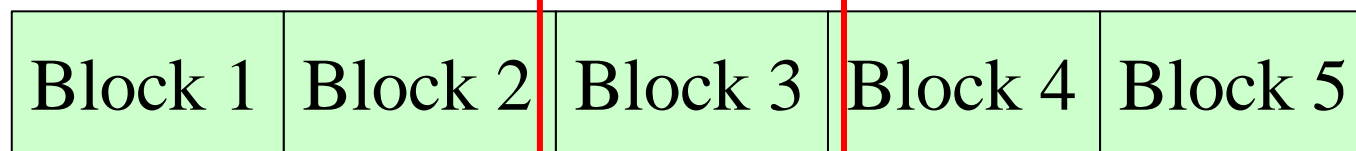
- ⌘ ブロッキング化が可能
- ⌘ 配列要素の再利用間隔は一定
SCMAで効率的な prefetch が可能
- ⌘ 配列ごとの再利用性の差は小さい
再利用性に基づいたデータ分類は困難
- データ転送より演算にかかる時間の方が大
先行ロードによって latency の隠蔽が有効

輻射輸送問題の最適化

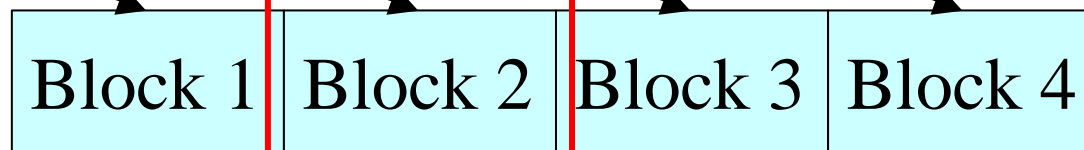
1. ブロッキング化
2. SCMAによるデータ流のパイプライン化

On-chip memory を3分割

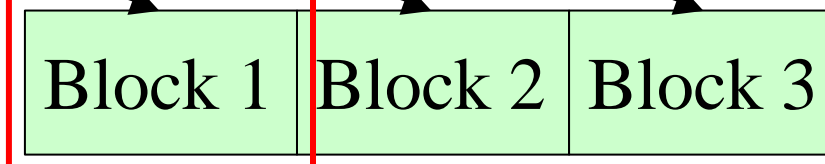
データの読込
page-load



演算



データの書き戻し
page-store



プログラムの作成

⌘ SCIMA プロセッサのクロックレベルシミュレータ

⊠ MIPS4 ISA(R10000) をベースとして SCIMA 専用命令を拡張

⊠ Super Scalar、Out-of-order

⌘ バイナリの生成

⊠ MIPSpro C Compiler でアセンブラ出力

⊠ 疑似関数を SCIMA 命令に変換するスクリプト

⊠ 拡張されたレジスタに対応して register spill を除去するスクリプト

大河原'00

評価 (問題規模)

⌘ 問題サイズを空間格子数 32

⌘ 以下の 2種のアーキテクチャを想定比較

(1) 128KB 4-way cache

(2) 8KB 2-way cache + 128KB On-chip mem

⌘ 実行するプログラム

1. Cache Only (1)

2. Scima Opt. 1 ((2)先行ロードなし)

3. Scima Opt. 2 ((2)先行ロードあり)

評価 (シミュレーション環境)

SCIM-0.4.2 を使用

動作周波数 : 内部 2GHz、外部 1GHz

同時発行命令数 :

整数演算 2、浮動小数点演算 2~8

load/store 命令 **cache 2、on-chip mem. 4**

レジスタ数 : **整数32本、浮動小数 64本**

メモリアクセス :

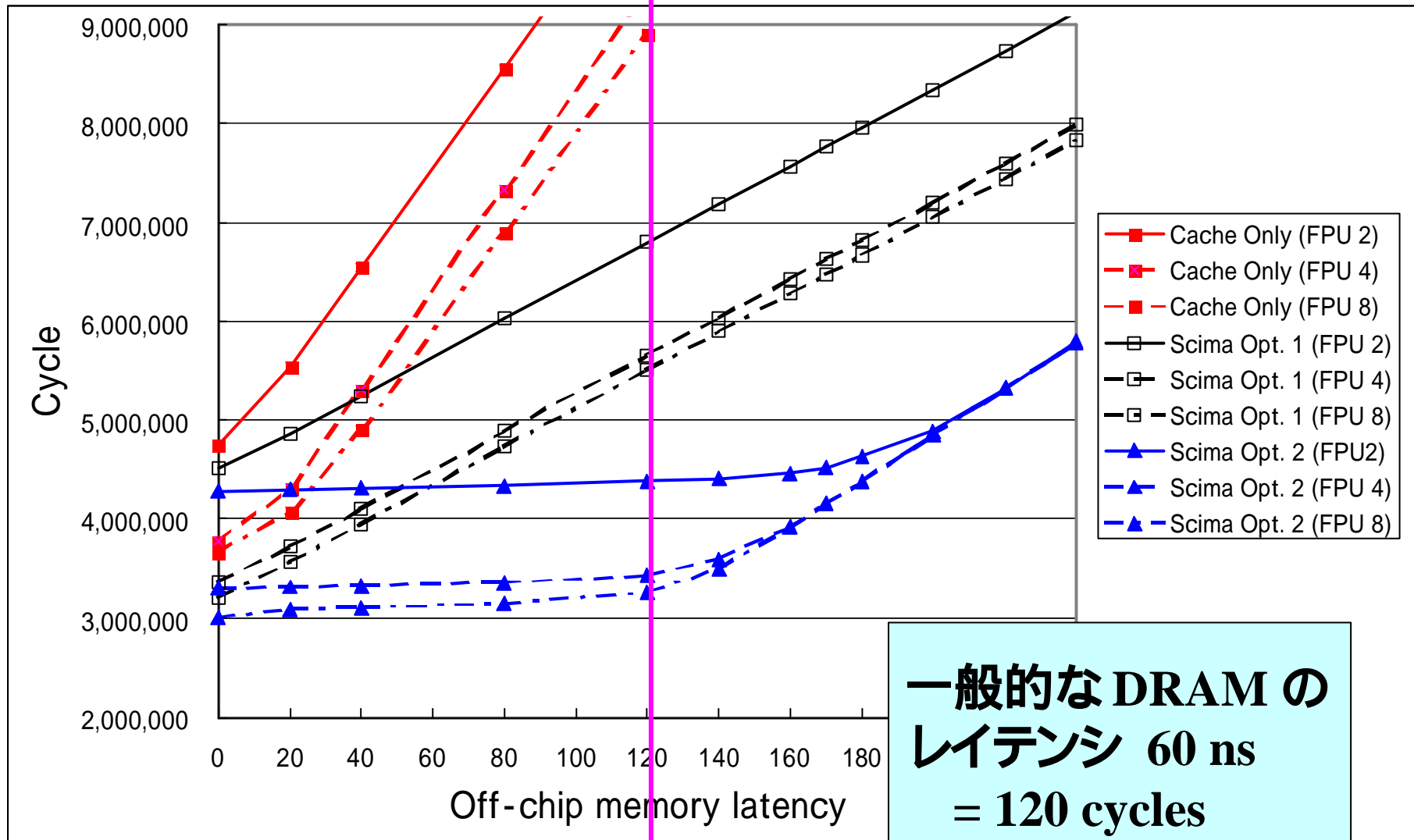
バンド幅 16Bytes/cycle

レイテンシ 0 ~ 240 cycles

throughput

on-chip:off-chip = 3:1

結果 (実行サイクルの比較1)



一般的なDRAMの
レイテンシ 60 ns
= 120 cycles

結果の考察

page-load による先行ロードの効果は非常に高い。

Off-chip memory latencyに対する耐性

- 0 ~ 120 cyclesの間では高い耐性
- それ以上では latency の影響が無視できない
ブロックサイズが大きくなると改善

Scima Opt. 1(先行ロードなし)においても、Cache 以上の性能が出ている。

転送単位が大きいことによる効果

結果の考察



- × FPU ユニットを増やした場合の性能向上が鈍い。
 - FPU ユニットが増加し演算サイクル数が減少
On-chip memory L/S ユニット数が相対的に不足

SCIMA コンパイラ

これまで

☒ プログラマが SCIMA のディレクティブ (page

1. SCIMA コンパイラのベースとなる newcc の解説

2. SCIMA コンパイラ最適化手法 (Page-load/store pipelining) の提案と評価

本稿では

☒ 一定の SCIMA 最適化が施されたコードからの最適化手法を提案。C コンパイラ上に実装。

一般的なCコンパイラの構成

⌘ Frontend

- ☑ 字句解析、構文解析

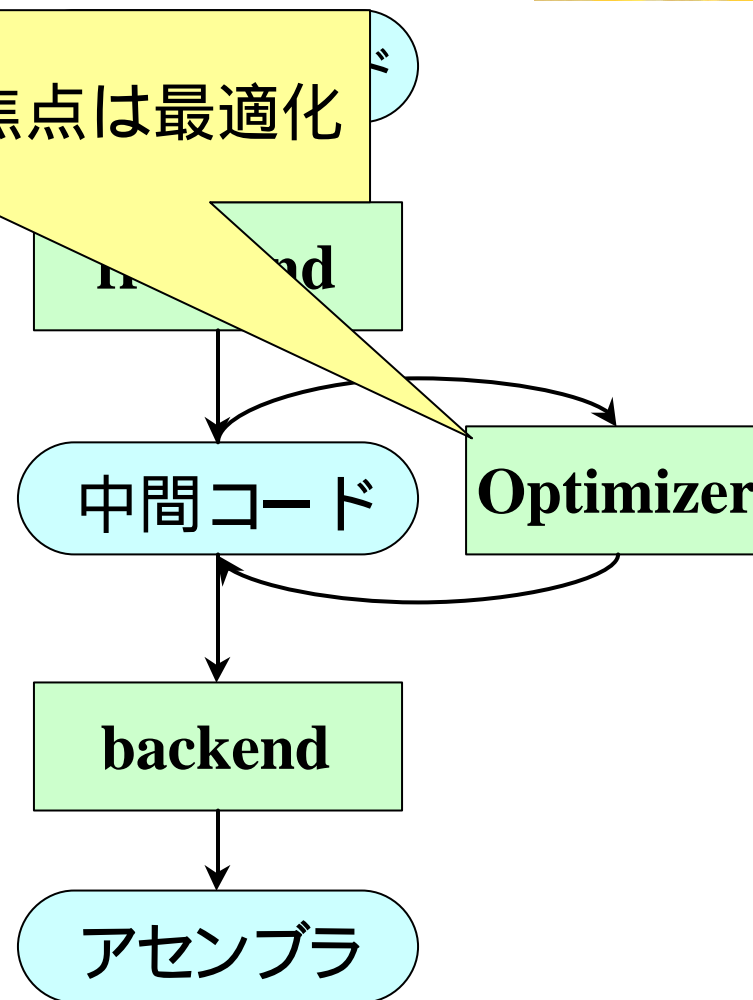
現代のコンパイラの焦点は最適化

⌘ Backend

- ☑ レジスタ割当、マシン語への変換

⌘ Optimizer

- ☑ 中間コードに対して最適化
中間コードの形式は処理系に固有 cf. RTL(gcc)



最適化とは

一般的なCコンパイラ
は命令レベルのみ

命令レベル

- ⊠レジスタ割当
- ⊠高速/特殊な命令の使用
- ⊠命令レベル並列化 (ILP)
- ⊠冗長な演算の除去

スレッドレベル

- ⊠スレッドレベル並列化 (TLP)

データ配置レベル

- ⊠キャッシュ最適化
- ⊠データ分散

SCIMA 最適化は
データ配置レベル

局所的・具象的
人間が苦手

大局的・抽象的
コンパイラが苦手

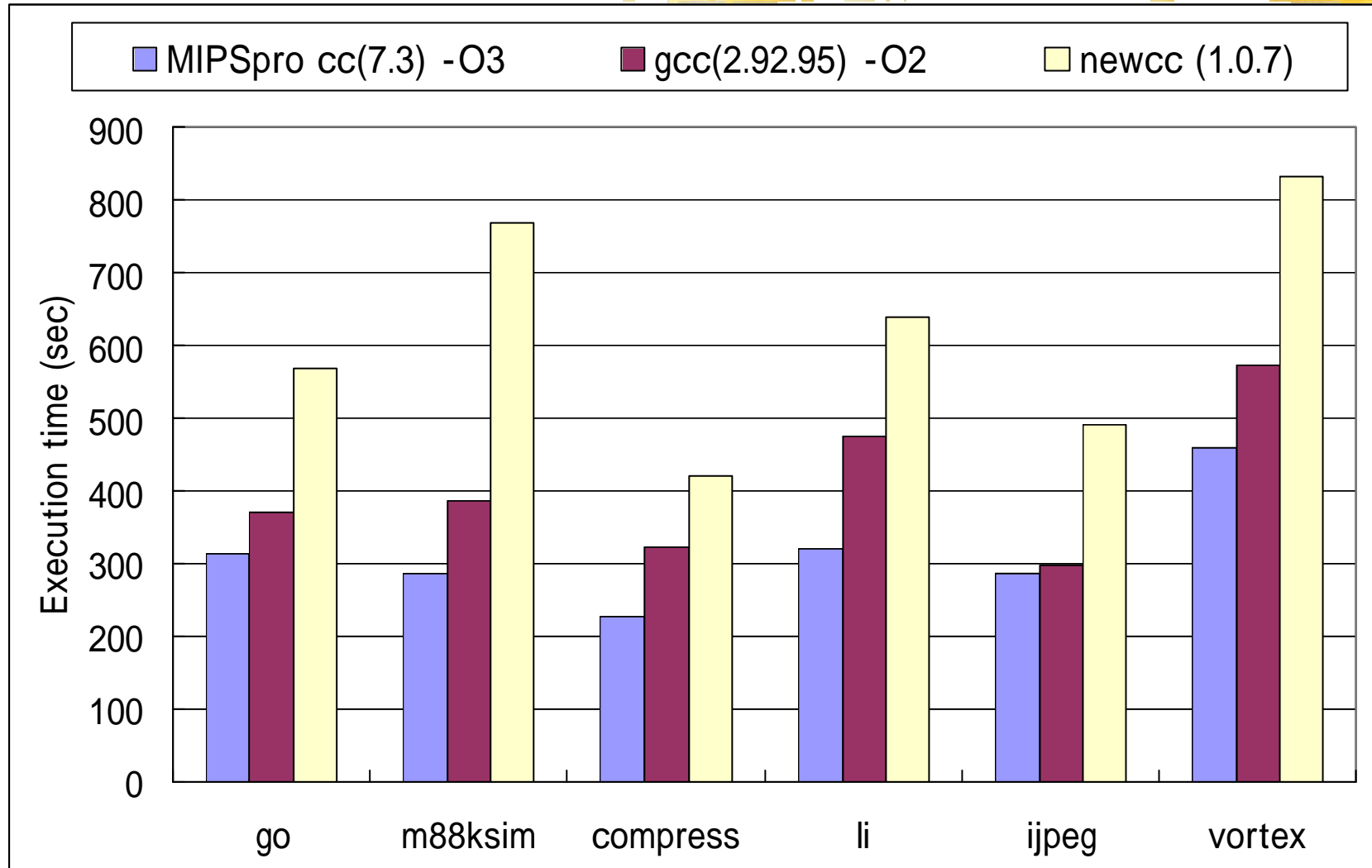
最適化 C コンパイラ newcc

SCIMA 最適化 C コンパイラの実装にあたり
newcc をベースとする。

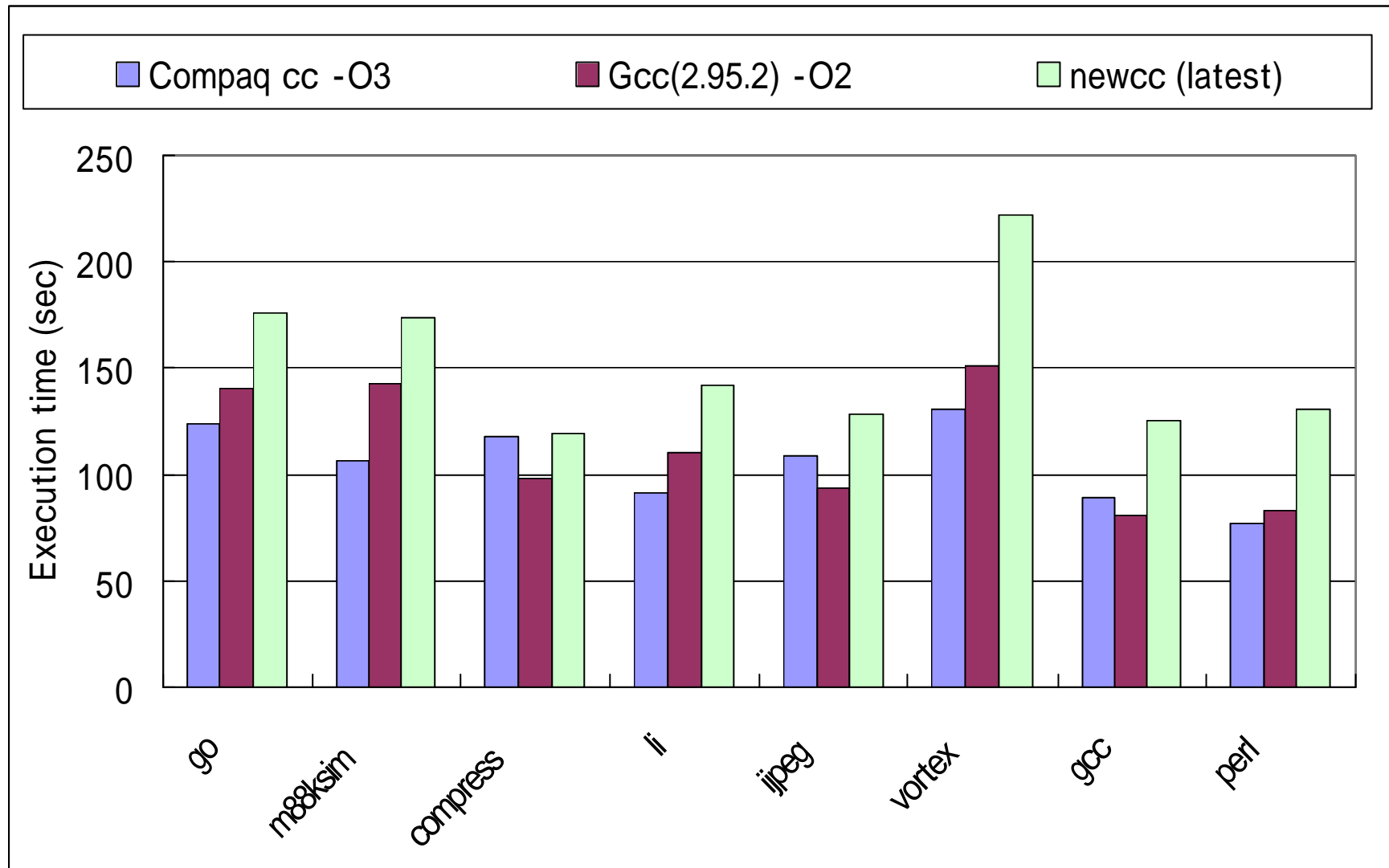
- ⊡ 富士通研で研究・開発 (SPARC版)
(複層構造の中間言語を持つ研究用コンパイラ)
- ⊡ MIPS4 R10000/ Alpha 21264 へ移植
- ⊡ 一般的な命令レベル最適化の実装
(backend と optimizer が同一)
- ⊡ SCIMA ディレクティブに対応
(疑似命令を含んだ MIPS4 アセンブラを出力可能)

性能評価 - SPECint95

(SGI Origin 2000 / R10000)



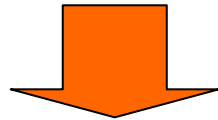
性能評価 - SPECint95 (Alpha 21264)



提案するSCIMA最適化

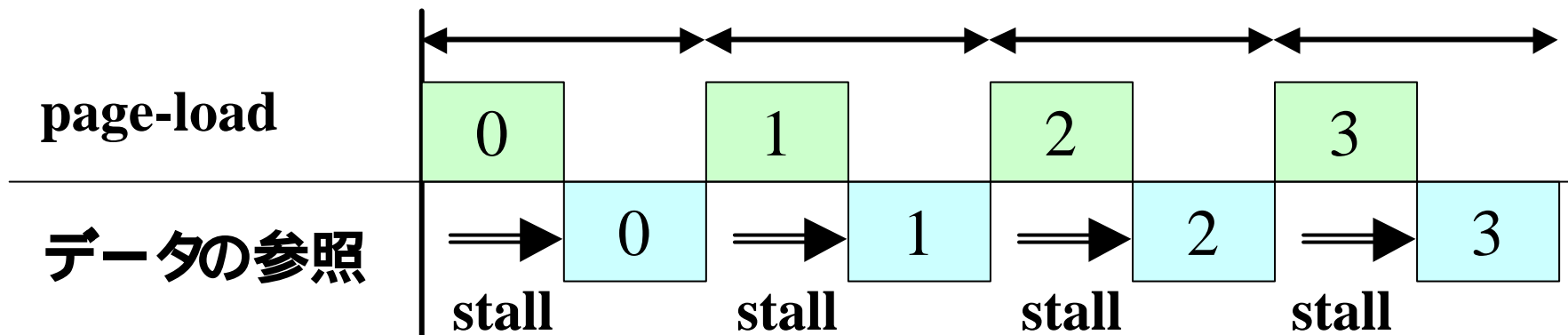
Page-load/store pipelining

page-load/store を単純に使っただけでは先行ロードの効果がない。



```
for(i=0; i < N; i++) {  
    page-load(i番目);  
    // i番目を参照  
}
```

page-l/s と load/store 命令が接近するとロック (順序保証機構)



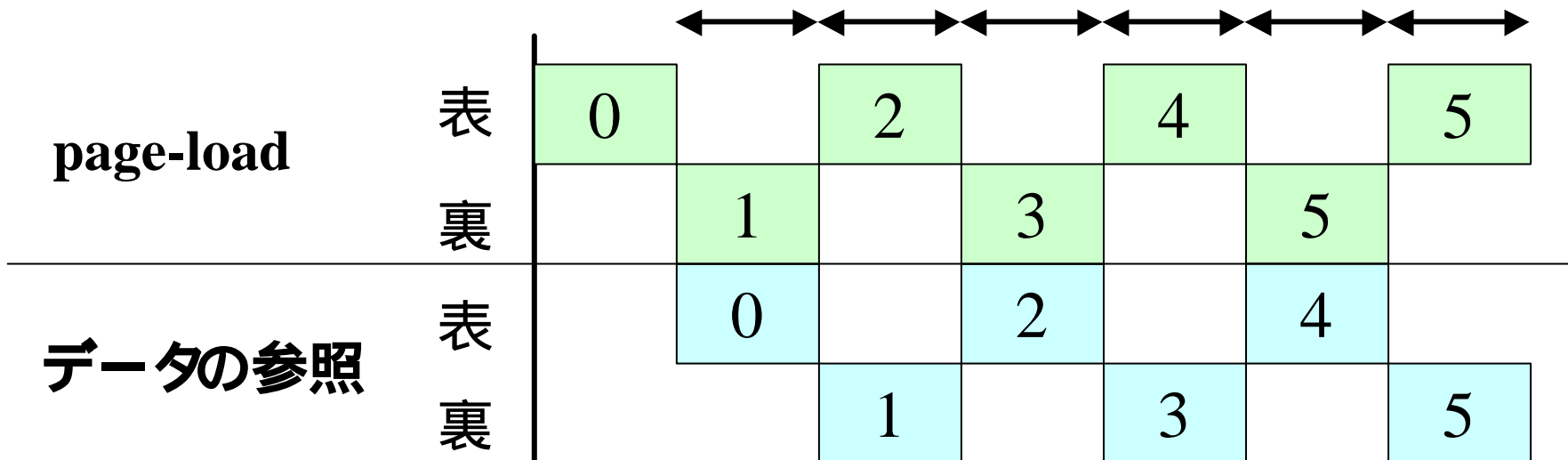
提案するSCIMA最適化

Page-load/store pipelining

Double buffering 化

データ流のパイプライン化

```
page-load( 0 番を表に );  
for(i=0; i < N; i+=2 ) {  
    page-load( i+1番を裏に );  
    // i 番目(表) を参照  
    page-load( i+2番を表に );  
    // i+1 番目(裏) を参照  
}
```



提案するSCIMA最適化

Page-load/store pipelining

データ依存解析

__Double buffer 化可能の判定

- page-load/store、load/store の
アドレスがループ誘導変数の 1次式
$$OnChip[x+y*N]$$

- ループの繰り返しを跨ぐ依存の解析

検出 { 同一の off-chip memory の重複転送
異なる iteration に属する on-chip memory のアクセス

```
for(i=0; i < N; i++) {  
    page-load(i 番目);  
    // i 番目を参照  
}
```

Page-load/store pipelining の評価

評価方法

Normal	オリジナル (page-l/s が入っただけ)
Compiler	コンパイラ最適化
Ideal	page-l/s pipelining の理論限界
Handwrite	プログラマによる最適化 (比較)

評価プログラム

- ☑ 行列積 (blocking) \ 行列積 (non-blocking) \ Q
R分解1、QR分解 2、LU分解、輻射輸送計算

評価 (シミュレーション環境)

SCIM-0.5.2 を使用

動作周波数 : 内部 2GHz、外部 1GHz

同時発行命令数 :

整数演算 2、浮動小数点演算 4

load/store 命令 2 (on-chip, cache 共用)

レジスタ数 : 整数32本、浮動小数 64本

メモリアクセス :

バンド幅 8Bytes/cycle

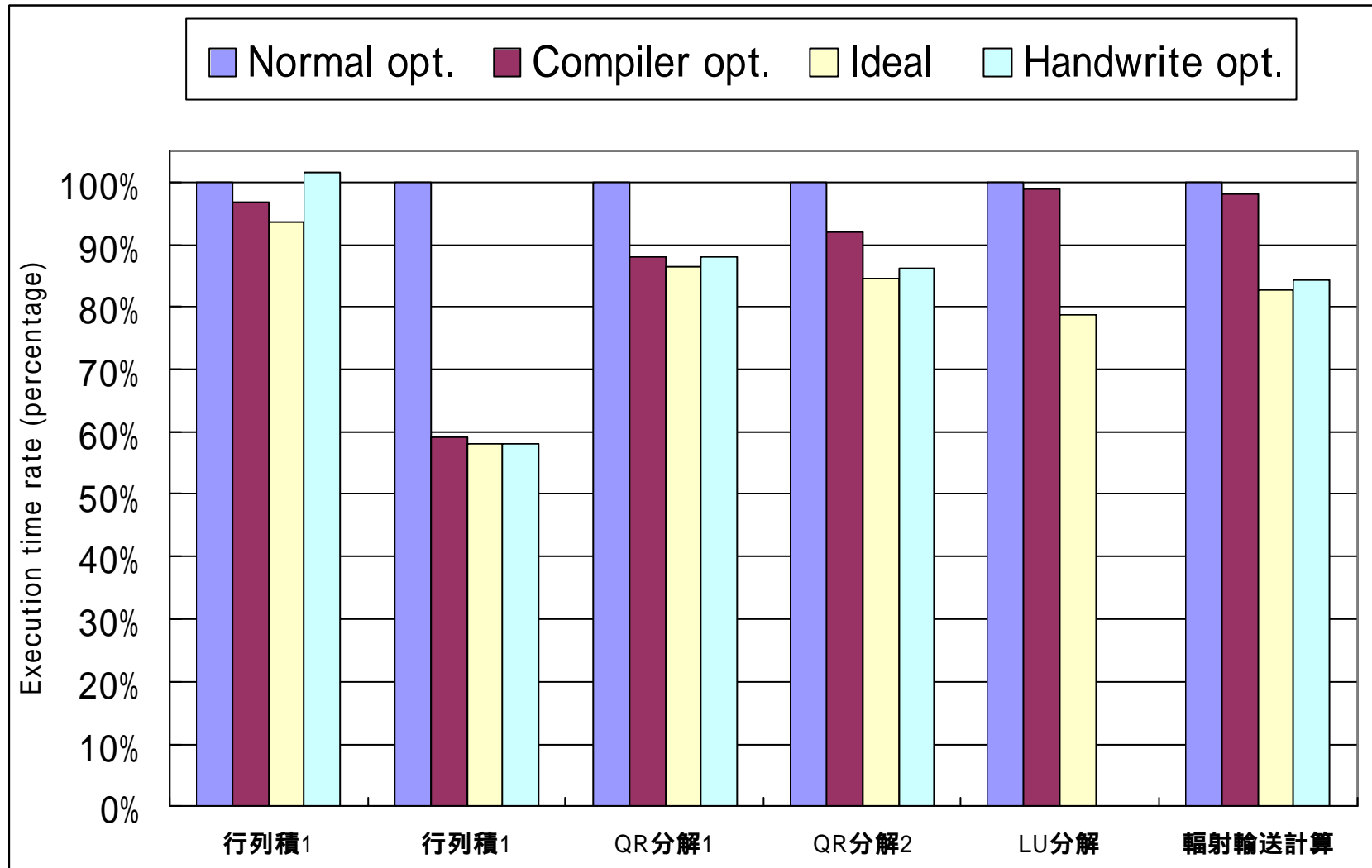
レイテンシ 0 ~ 200 cycles

throughput

on-chip:off-chip = 2:1

結果

(off-chip memory latency 120cycles)



考察

行列積 (blocking) , 行列積 (non-blocking)

⊡最適化が理論限界に近い

輻射輸送計算

⊡性能向上が小さい

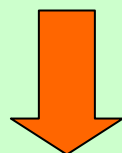
同一ページへのアクセスで競合

ハードウェア制御

- Color Queue、ページ単位の管理

ソフトウェア制御

- Weakly Ordered
- **wait 命令** (page-1/s の完了するまで spin lock)

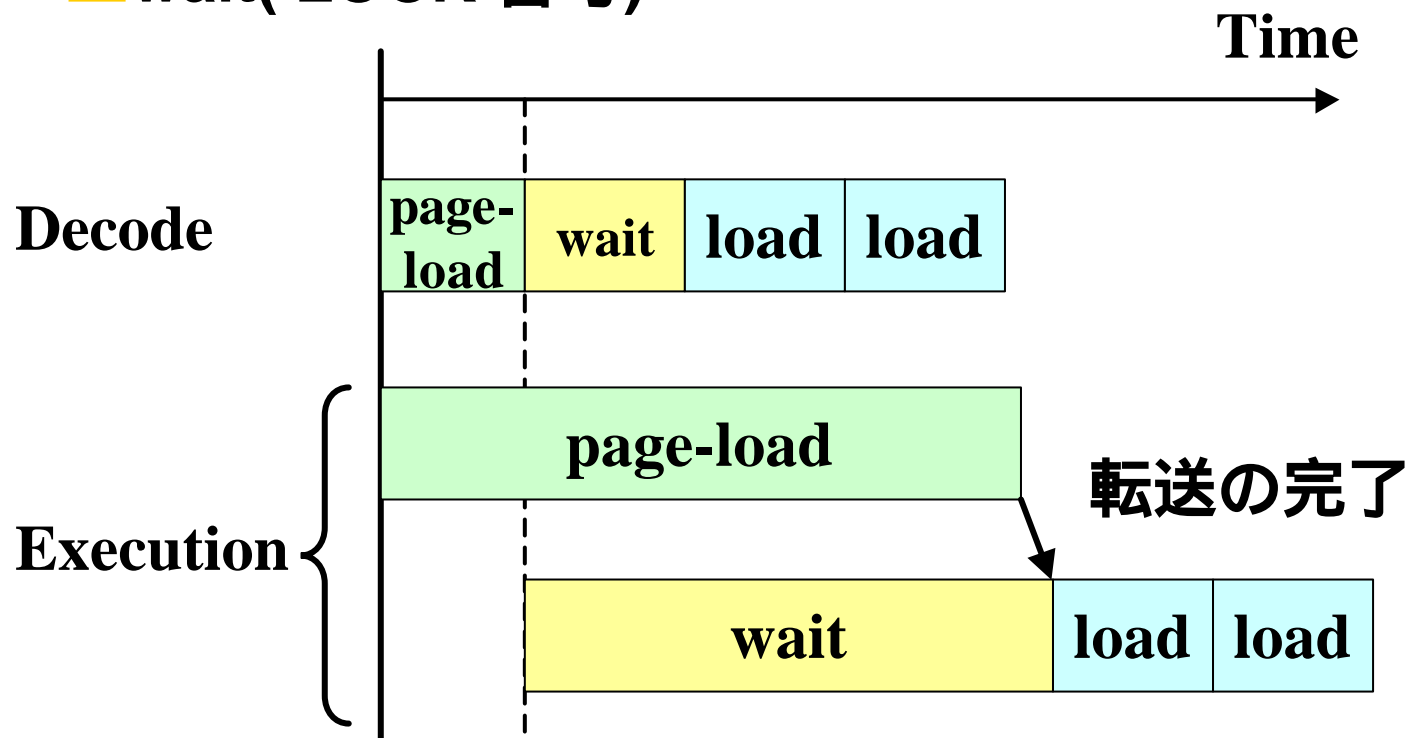


同期のソフトウェア制御

SCIMA 命令

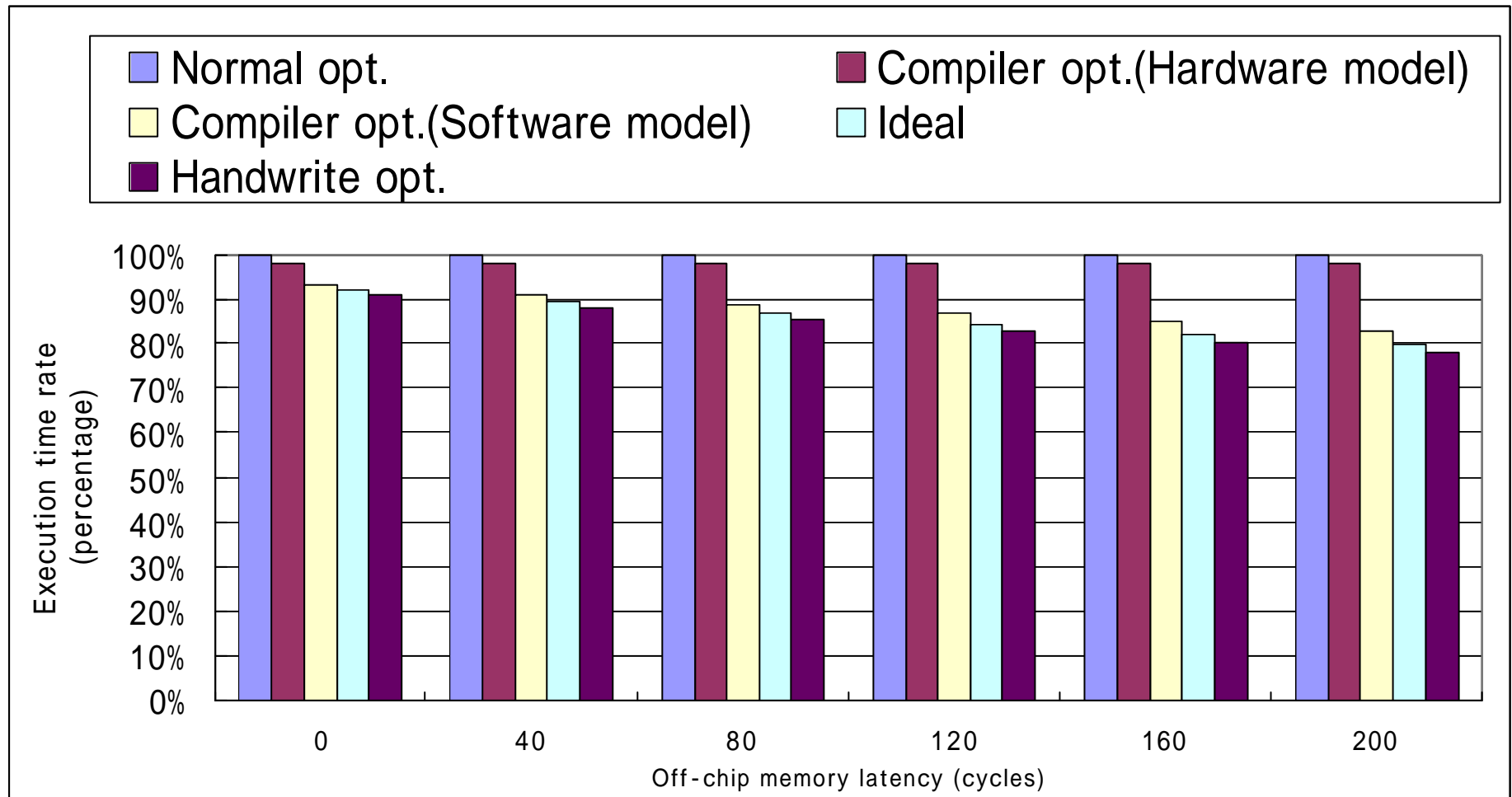
⏏ pload(.....,,,,,, LOCK 番号)

⏏ wait(LOCK 番号)



結果

輻射輸送計算での Hardware model と Software model の比較



まとめ



輻射輸送問題を SCIMA に最適化・評価

SCIMA アーキテクチャの有効性を示した。

SCIMA 最適化 C コンパイラ

page-load/store pipelining を実装

データの流を意識せずにプログラム可能

今後の課題と展望(1)



一般的な最適化の強化

- ループ最適化
- 命令スケジューリング

HPC 最適化

- キャッシュ最適化 (キャッシュブロッキング等)
- Pointer aliasing analysis

SCIMA 最適化

- On-chip memory の使い方をコンパイラが自動的に決定
- On-chip memory を使った spill code の除去

今後の課題と展望(2)



他言語への対応

- newcc backend に対応する C++、Java(Native Compiler)、FORTRAN frontend の開発

他アーキテクチャへの対応

- CMP、VLDP