次世代チップ・マルチプロセッサのメモリ・アーキテクチャ

林, 徹生
九州大学大学院システム情報科学府

出版情報：SLRC プレゼンテーション, 2007-07-25
バージョン：
権利関係：

次世代チップ・マルチプロセッサのメモリ・アーキテクチャ

国立大学法人九州大学
大学院システム情報科学府
修士課程2年
林　徹生

発表内容

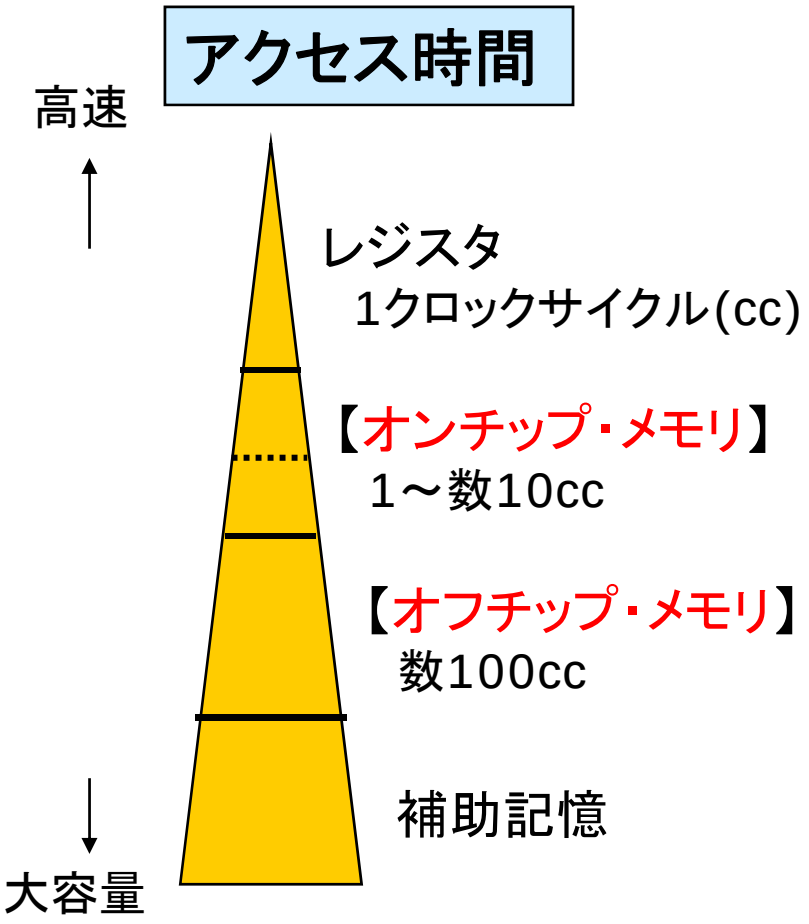
- CMPの普及、将来はメニーコアへ？
  - オンチップ・メモリはキャッシュ・メモリからソフトウェア制御可能なメモリへ移行？
  - より顕著に？　メモリ・ウォール問題。
- 暇なプロセッサ・コアをサポートに出そう(提案手法)！
  - ユーザ(プログラマ)の負担は増えるの？
  - どんなメカニズムで動作するの？
  - 評価、始めました。
- 終わりに

CMPの普及、将来はメニーコアへ？

- 1チップ上に複数のプロセッサコアを搭載
- スレッドレベル並列性(TLP)により性能向上
  - 汎用
    - Dual Core(2コア)が主流
    - 4コアのプロセッサへ移行
  - 特定用途
    - 8コア：Cell(Sony,Toshiba,IBM)、Niagara2(Sun)
    - 16コア：Raw(MIT)
    - 80コア：Intelのプロセッサ
    - 96コア：CSX600(Clear Speed)
- 今後もコア数は増加と予想

コア内部にソフトウェア制御可能なメモリ(以降、ローカル・メモリと呼ぶ)を持つ(Rawはキャッシュも持つ)。

より顕著に？　メモリ・ウォール問題



- オフチップ・メモリへのアクセス時間が計算機の性能ボトルネックになることが多い

⇒『階層メモリ構造』により、アクセス回数の削減、アクセス時間の隠蔽を図っている

今後

コア数増加に伴い、演算器は増加。一方、メモリサイズはあまり増加しないor減少と予想。

メモリ性能がより重要に！

メモリ性能を向上したい！

- ある時間において、未使用コア or 暇なコアが存在
  - 並列性が低いプログラム
  - 各コアでワーキングセットサイズに大きな差

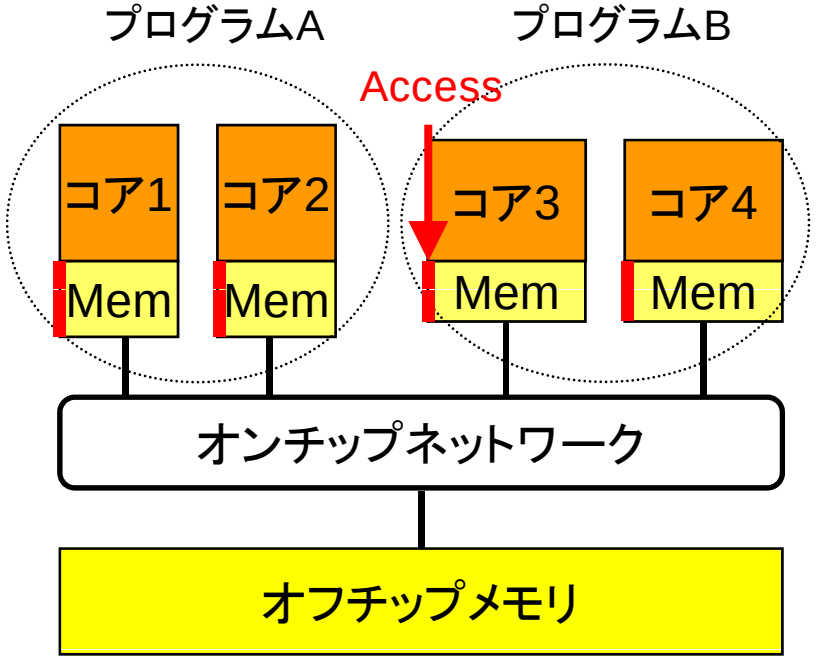
- 特にローカル・メモリ(＝ソフトウェア制御可)に着目
  - キャッシュ(ハードウェア制御)には様々な手法

未使用コアのローカル・メモリを階層メモリとして利用して性能向上を図れないか？

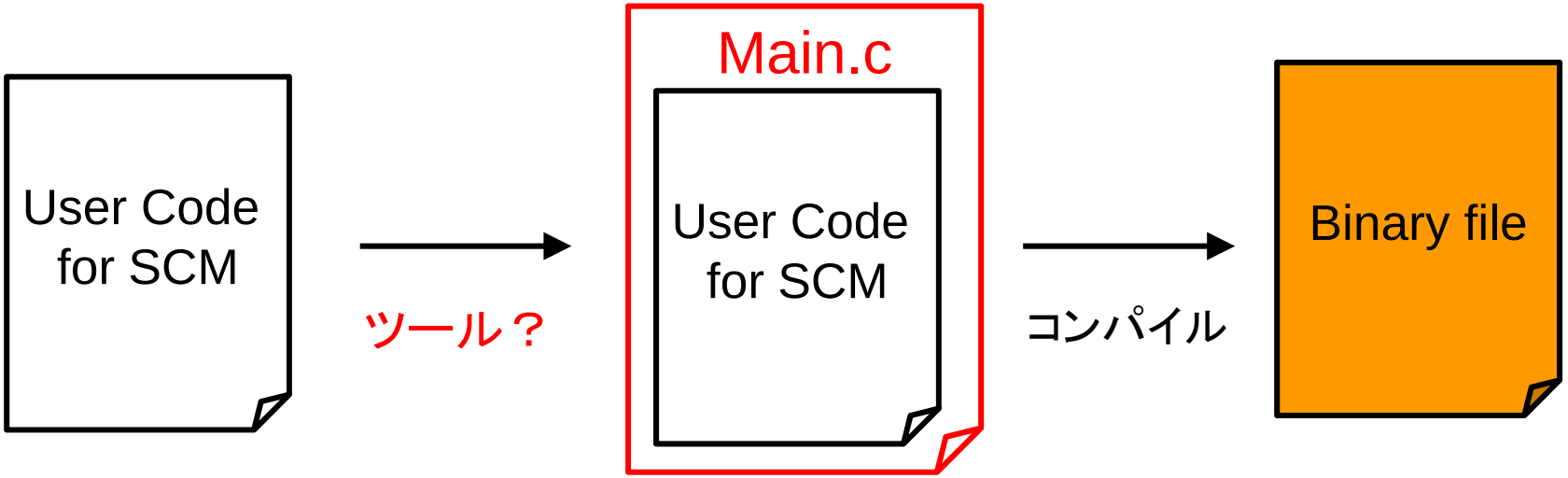
想定する環境

大前提： 他コアへのアクセス時間 ＜ オフチップへのアクセス時間

- プロセッサ
  - 各コアのローカル・メモリを物理アドレスで参照可能
  - ローカル・メモリはソフトウェア制御可能
- アプリケーション・プログラム
  - 複数の並列化されたプログラムを処理
  - 1つの並列化プログラムは同種のコアで処理



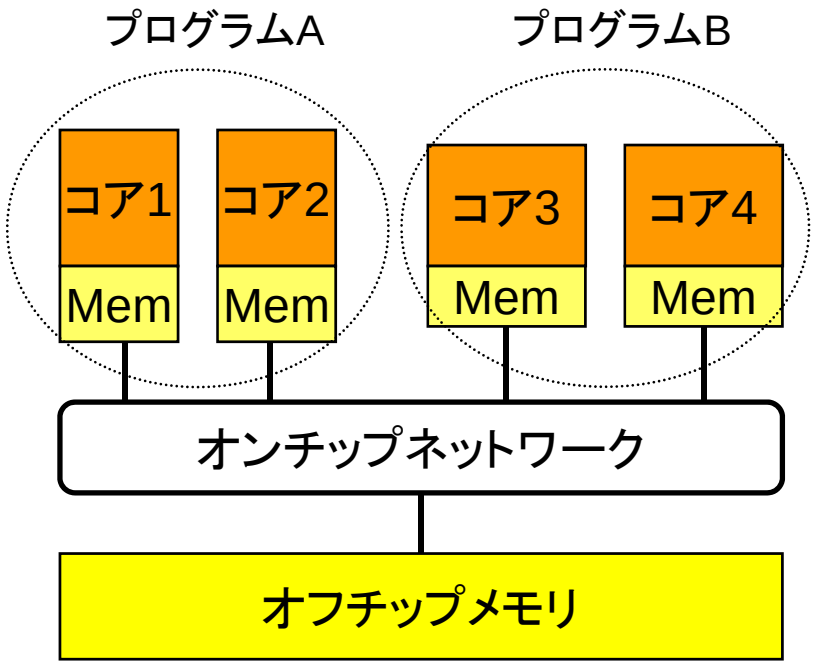
ユーザ視点～プログラマの負担は？～




```
Main(Base_address){
        _GET(offset1);
        func_a();
        _PUT(offset2);
}
```

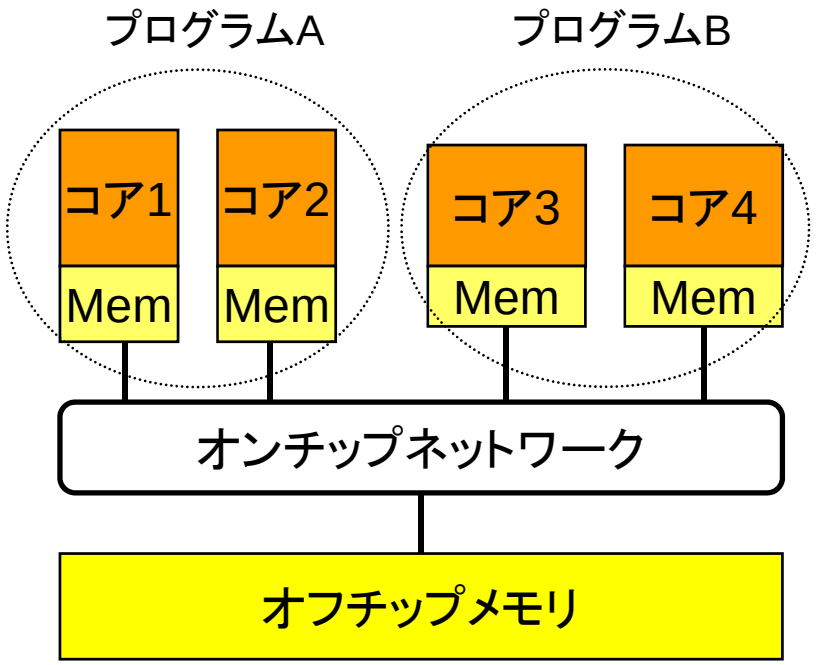
```
Boot_main(Base_address){
        copy_addr=Base_address
        Main(copy_addr);
}
when interrupt
        copy_addr = other_addr
```

メカニズム　～メモリ貸与開始～



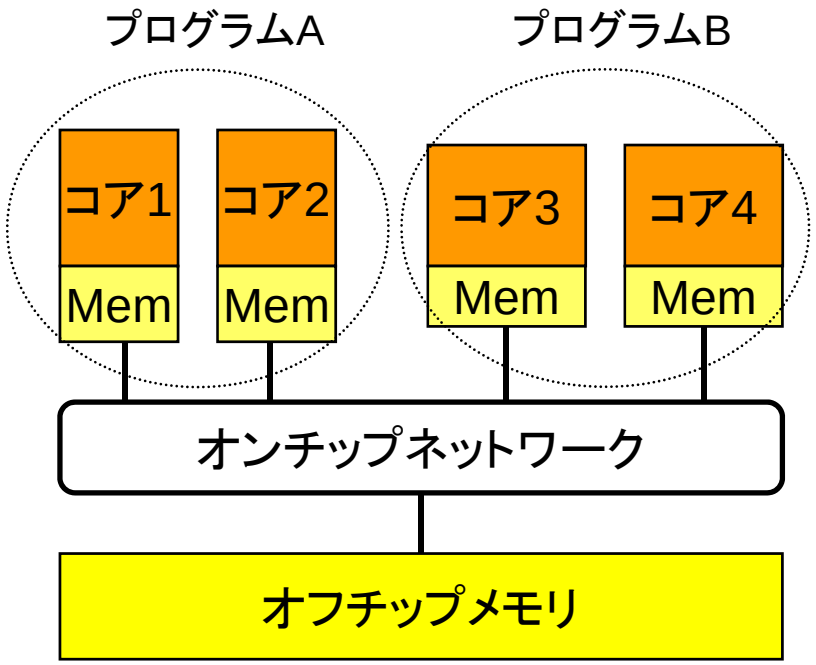
1. コア1,2が終了
2. コア1,2からコア3,4へ終了の合図
3. コア3,4がコア1,2のローカル・メモリを使用開始

メカニズム　～メモリ貸与中～



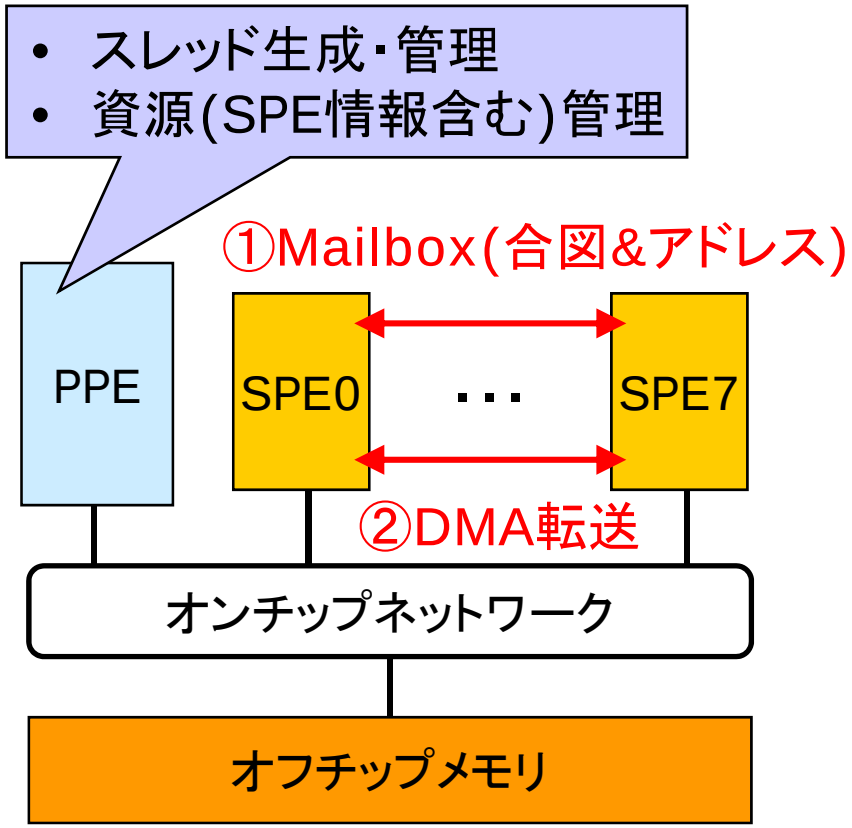
- コア3,4からオフチップへのアクセスをコア1,2のローカル・メモリへ
  - 読み込みの一部(データがない時)はオフチップへアクセス
- コア1,2は書き込まれたデータをオフチップへと書き戻す

メカニズム　～メモリ返却～



1. コア1,2は使用不可とコア3,4に合図
2. 書き込み処理(未完了のもの)
3. コア1,2はプログラムAを処理

実装 on Cell Broadband Engine



1. PPEが
   - スレッドを生成
   - SPEの物理アドレス情報生成

2. SPEが(書込み時)
   0. 各SPEの物理アドレスGET
   1. データの最後にオフチップ・メモリのアドレスを付与
   2. コアBのMailboxにコアAの物理アドレスを送信
   3. コアBがコアAのデータをDMA転送して自コアに格納
   4. オフチップ・メモリにデータ転送

評価、始めました。

- 行列積の計算
- 通信を隠蔽できない(＝bufferingできないと仮定)
- SPEのローカル・ストアは16Kbyteと仮定
- 提案手法におけるオーバーヘッドは無視

メモリサイズ / 時間(cycle)	16K	256K	Speed Up
実行時間		5,340,357	
通信待ち時間		37,322	
通信の割合(%)		0.70	

自コアの他に15コア
使用できる事と同義

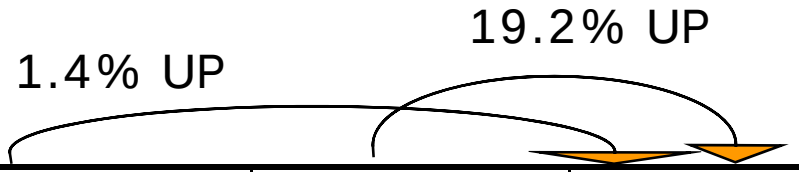
終わりに

- 未使用コアを活用した性能改善手法を提案
  - メモリ・ウォール問題に着目
  - ソフトウェア制御可能なローカル・メモリを対象
  - 【結果について】
- 【課題等？】

ご清聴ありがとうございました

Back Slides

Double Buffering/Inline適用時



メモリサイズ / 時間(cycle)	16K/inline/ no stall	16K/inline/ stall	16K/stall	256K/inline/ stall
実行時間	5,139459	5,415,231	6,612,327	5,340,357
if(BRmiss,DStall=½)	4,494869	4,683,464	5,340,004	4,512,452
通信待ち時間	34,176	87,378	87,376	37,322
通信の割合(%)	0.66	1.62	1.32	0.70
if(BRmiss,DStall=½)	0.76	1.87	1.64	0.83

BRmiss = Branch miss stall
DStall = Dependency stall

inline: inline展開
stall: Double Bufferingなし