SCIMAアーキテクチャと性能評価

- SCIMAアーキテクチャの概要 -

中村 宏

東京大学先端科学技術研究センター

nakamura@hal.rcast.u-tokyo.ac.jp
nakamura@acm.org

第一部：SCIMAアーキテクチャと性能評価

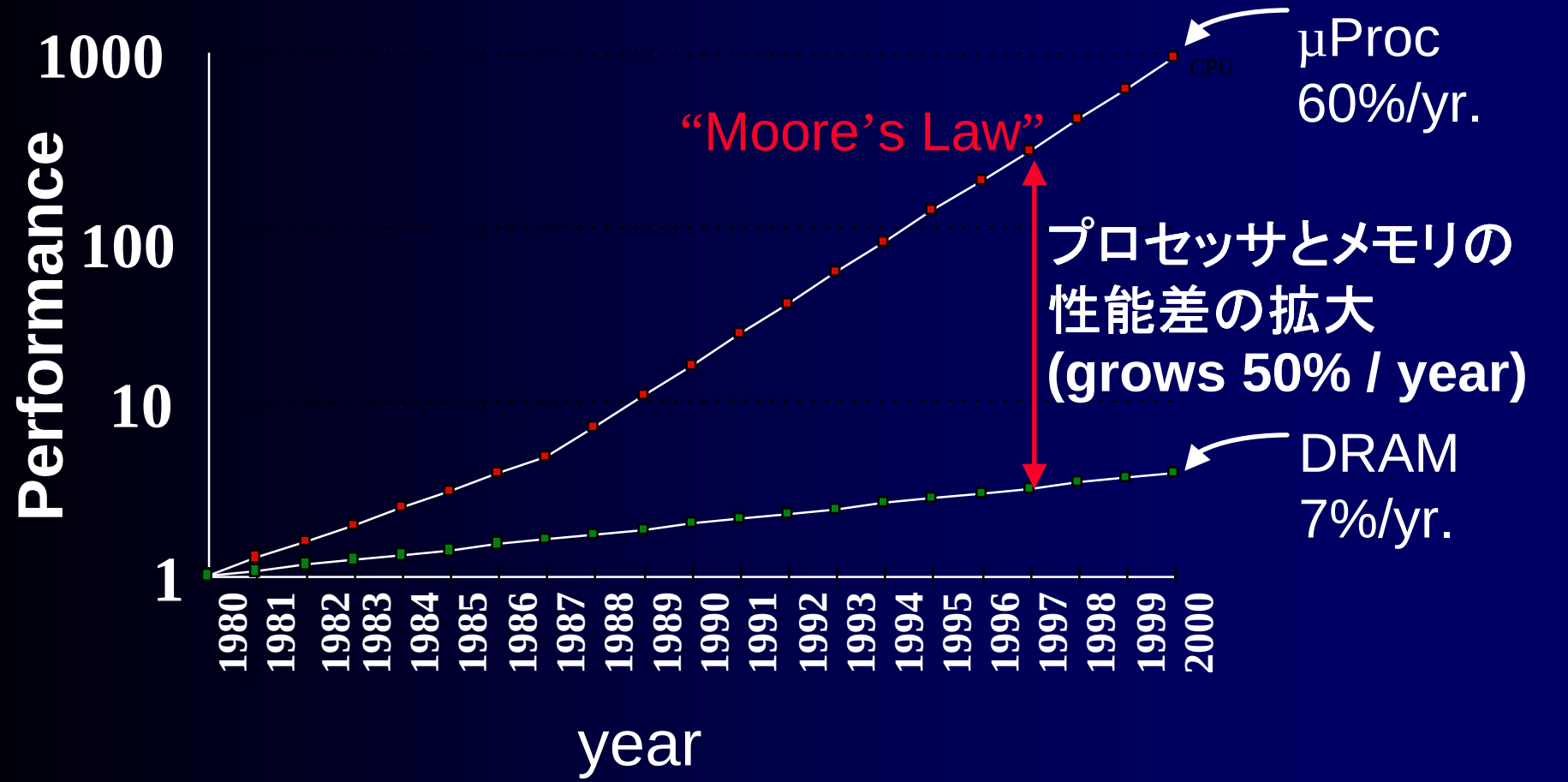
講演の流れ

- SCIMAアーキテクチャの概要　（東大: 中村宏）
- NASPBを用いたSCIMAの評価　（東大: 岩本貢 M2）
- 第一物性原理計算の高速化 –現状報告–
  （東大:大根田拓 M1）
- QCD計算と宇宙流体力学計算の高速化
  （東大:近藤正章 D1、宮崎隆志 B4）
- 次世代相互結合網のためのネットワークシミュレータ
  （筑波大: 大岩栄一郎 M1）

メモリ混載型プロセッサアーキテクチャ – なぜ新しいアーキテクチャなのか –

- プロセッサとメモリの性能差拡大

CPUとDRAMの性能格差



cited from: Fig 5.1. “Computer Architecture A Quantitative Approach (2nd Edition)” by J.Henessy and D.Patterson, Morgan Kaufmann (ISBN: 1-55860-329-8)

なぜ新しいアーキテクチャが必要か

- プロセッサとメモリの性能差拡大

- HPC（特に連続体）ではメモリ性能が処理能力を決める
  - long access latency
  - lack of throughput ← こちらのほうが本質的

→ レーテンシ隠蔽技術 (prefetch, preload) は、十分なメモリスループットが確保されてこそ効果を発揮

→ ベクトル計算機では、高価なインターリーブメモリを採用している

SIA Roadmap (1999 Edition)

	Year	2001	2003	2005	2008?	2011??
☺	Rule (nm)	100	80	65	45	30
☺	Tr./ch	220M	441M	882M	2.5G	7.0G
☹	Pins	3042	3042	3042	3840	4224
☹	signal I/O pins	1024	1024	1024	1280	1408
☺	Chip clock(Hz)	1.8G	2.5G	3.5G	6.0G	10G
☹	I/O clock (Hz)	1.5G	1.7G	2.0G	2.5G	3.0G
	Voltage (V)	1.5	1.5	1.2	0.9	0.6
	Power (W)	115	140	160	170	174

☺ チップあたりのトランジスタ数、チップ内のクロック周波数

→ ☺ チップ内部の性能は向上しつづける

☹ I/Oピン数、I/Oのクロック周波数

→ ☹ オフチップメモリのバンド幅は頭打ち

→ オフチップメモリのバンド幅を上げるのは極めて難しい

なぜ新しいアーキテクチャが必要か

- プロセッサとメモリの性能差拡大
- HPC（特に連続体）ではメモリ性能が処理能力を決める
  - long access latency
  - lack of throughput

  → オフチップメモリのバンド幅は限られている

- バンド幅を向上できるローカルメモリ（チップ上のメモリ）を有効活用することが必須

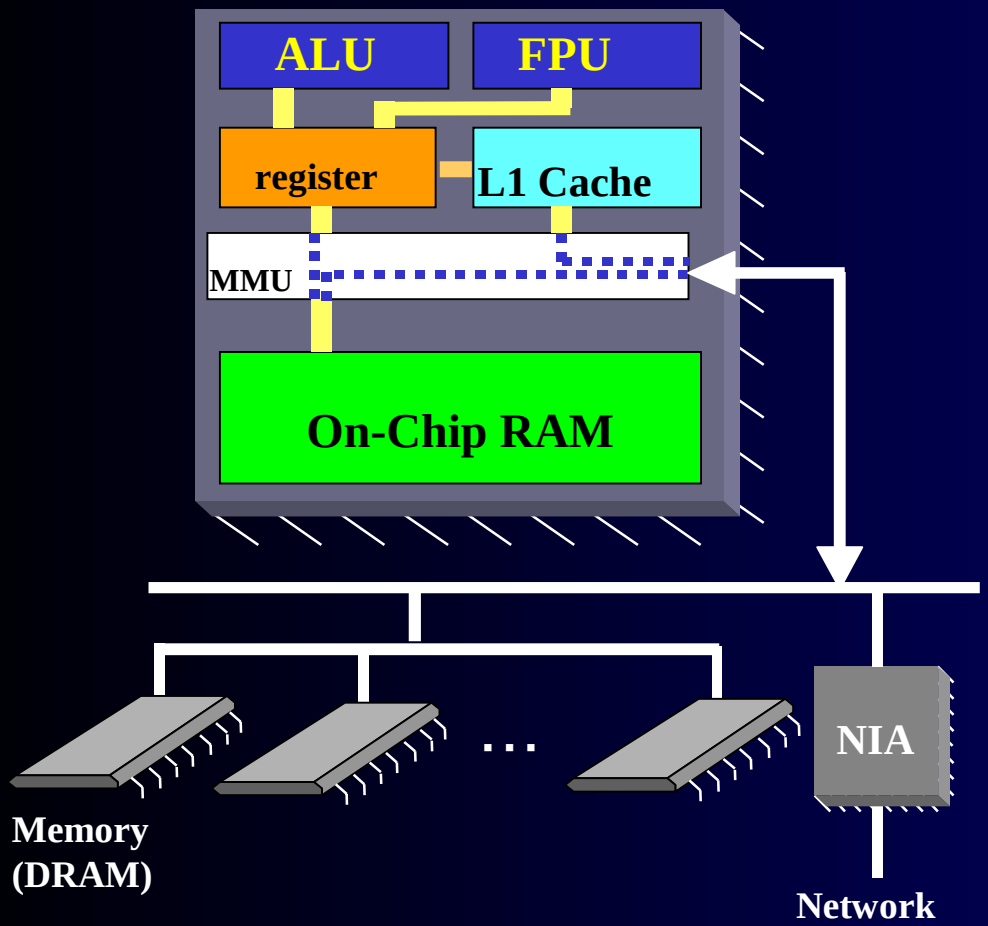
  オンチップメモリは非常に貴重な資源
- 通常のキャッシュはHPCではうまく利用できない

キャッシュの問題点

- データではデータのアロケーション／リプレースメントはハードウェア制御
  - 殆どのデータアクセスは規則的なのに、ラインコンフリクトによるキャッシュミス発生
    殆ど使われないデータが再利用性のあるデータを追い出す
- 転送サイズがラインサイズに固定
  - 連続アクセス：大粒度アクセスを行いたい
  - 非連続アクセス：ラインサイズ大→無駄なトラフィック

⇒ソフトウェア制御可能なメモリアーキテクチャの提案

SCIMA：（Software Controlled Integrated Memory Architecture）



SCIMAの概念図

チップ内：キャッシュ以外にオンチップメモリ(SRAM)を搭載

チップ外：HPCの大規模データセットを想定しチップ外にもオフチップメモリ(DRAM)を配置

キャッシュ vs. オンチップメモリ

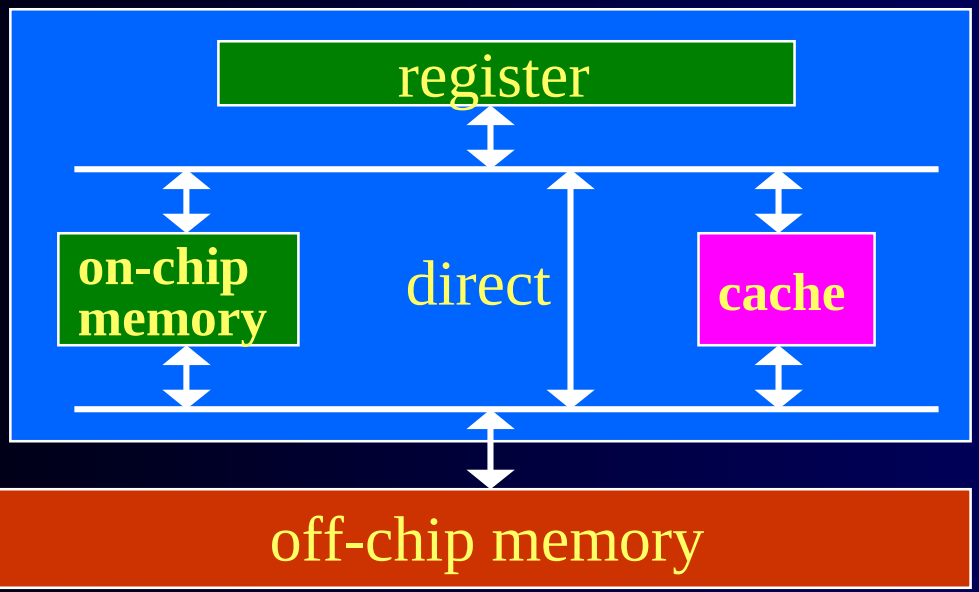
	cache	on-chip memory
control	hardware	software
transparency	○	×
user controllability	×	○

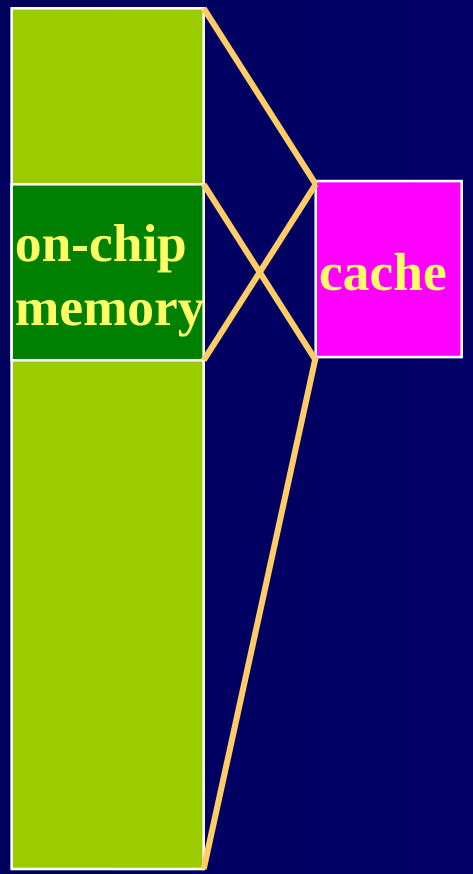
- 一般的には transparency が重要
- HPCでは、殆どのデータアクセスが規則的、かつ性能がもっとも重要

→ ユーザ（プログラマ・コンパイラ）に頑張ってもらって性能向上を目指す

メモリ階層

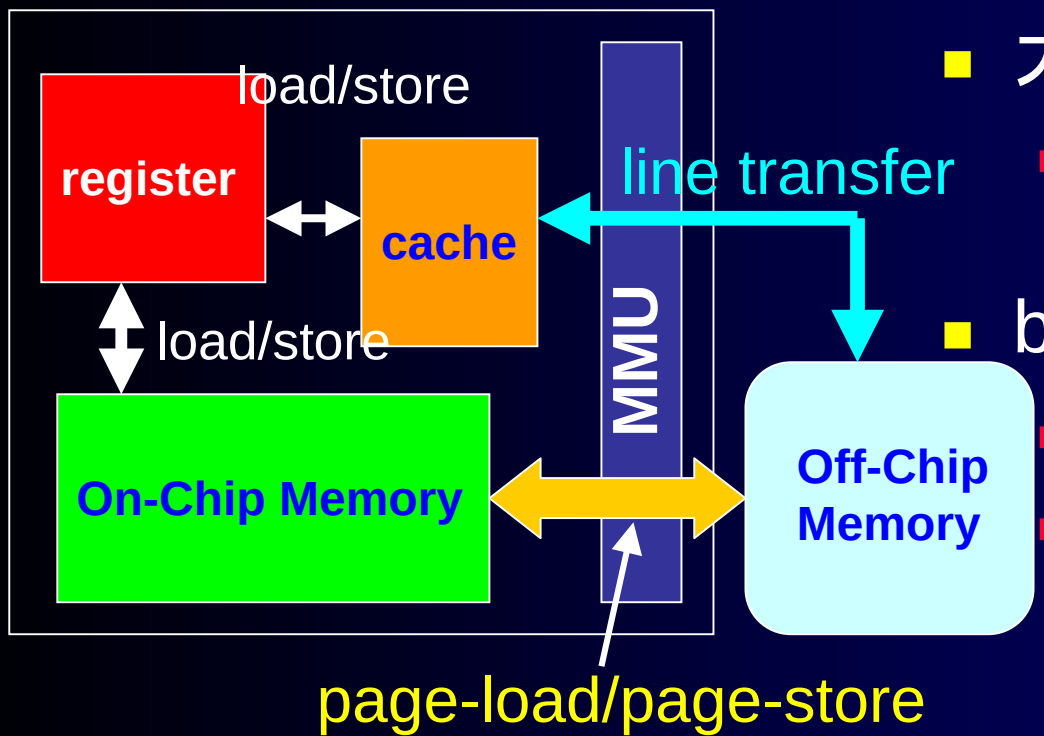
- オンチップメモリ
  - 論理アドレス空間の一部
  - キャッシングは不許可
  - キャッシュとの包含関係は無い





データ転送命令

- load / store
  - レジスタとキャッシュ、レジスタと オンチップメモリ
- page-load / page-store *New!*
  - オンチップメモリとオフチップメモリ



- 大粒度転送
  - レーテンシの影響を緩和 → 実行バンド幅拡大
- block stride transfer
  - 不要なデータ転送を排除
  - オフチップメモリバンド幅とオンチップメモリ領域を無駄にしない

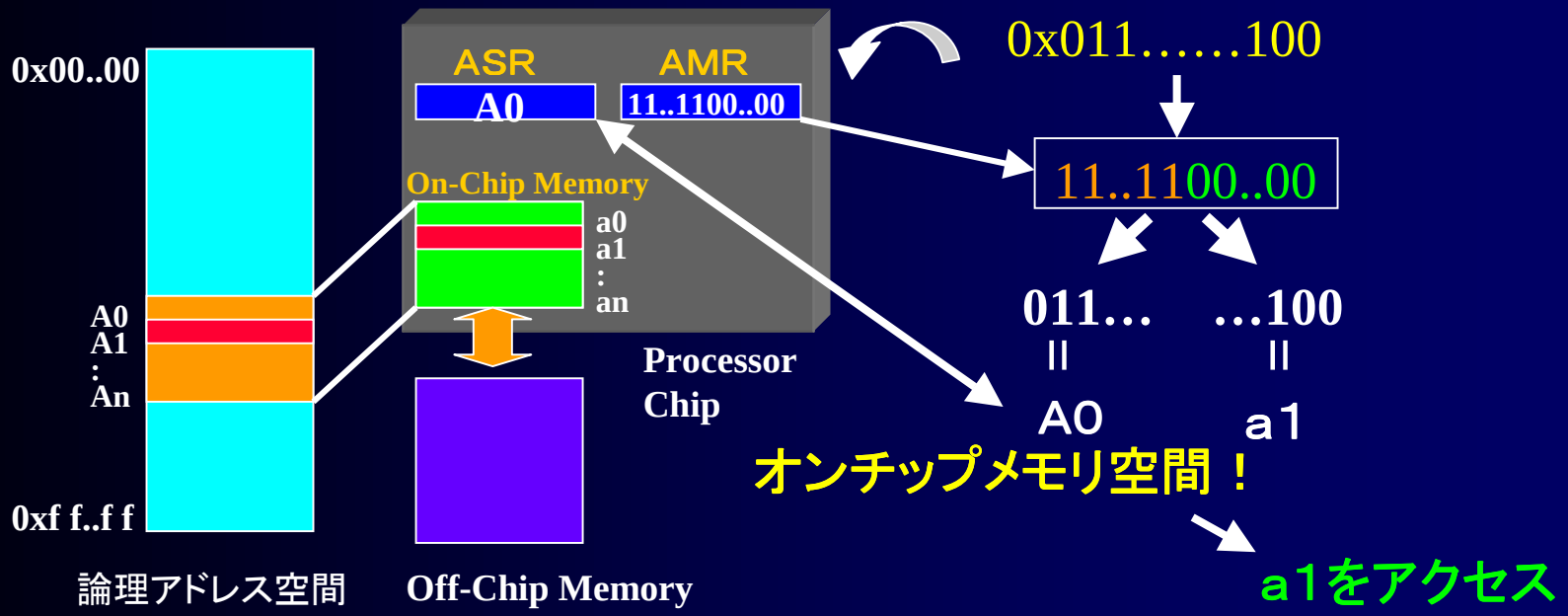
オンチップメモリの管理

特別なLargeページとして管理する

テーブルではなく、特別なレジスタで管理

On-Chip Address Start Register（ASR）：開始アドレス

On-Chip Address Mask Register（AMR）：容量



オンチップメモリとキャッシュの統合

- オンチップメモリとキャッシュの総容量には上限
  最適なオンチップメモリとキャッシュの容量比はアプリケーションに依存
- キャッシュの一部（または全部）をオンチップメモリとしても使えるようにする。変更の粒度は、way

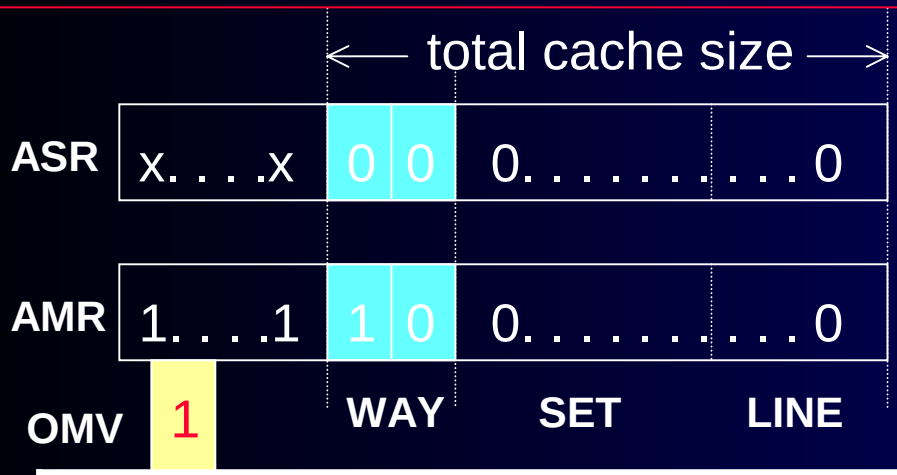
例） 4way associative cache(1way 8kB)の場合

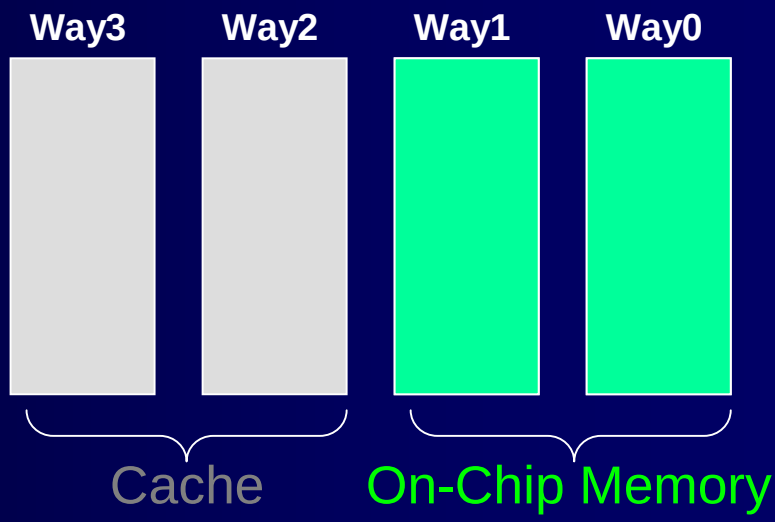
a. キャッシュ 32kB + オンチップメモリ 0kB
b. キャッシュ 24kB + オンチップメモリ 8kB
c. キャッシュ 16kB + オンチップメモリ 16kB
d. キャッシュ 0kB + オンチップメモリ 32kB

注：オンチップメモリ容量は2のべき乗であるという制約
→ キャッシュ 8kB + オンチップメモリ 24kB構成は無い

キャッシュとオンチップメモリの統合機構

e.g. 32KB, 4-way set associative cache





On-Chip Mem size	way bit of ASR	way bit of AMR	way for On-Chip Mem	OMV
32KB	00	00	way0,1,2,3	1
16KB	00 10	10	way0,1 way2,3	1
8KB	00 01 10 11	11	way0 way1 way2 wat3	1
0KB	--	--	N/A	0

possible configurations

キャッシュ/オンチップメモリ比は いつ変更できるのか

- プログラムの実行中いつでも可能 by software
- サイズ変更のオーバヘッドは大きい
  - キャッシュの中身を主記憶へ書き戻す
  - オンチップメモリの内容はユーザ責任で管理
- 複数の time consuming loop の間位が適当
  （後の講演では、プログラム途中での変更なし）

評価におけるソースコードの作成法

- 以下に提供する予約関数libraryを用いる
  - get_OnChipAddr()
    - オンチップメモリ領域を確保し、先頭アドレスを返す
  - p_load(*source, *desti, size, blocksize, stride)
  - p_store(*source, *desti, size, blocksize, stride)
    - 引数としてブロックストライド転送に必要な情報を与える
      （転送元、転送先、総転送サイズ、転送ブロックサイズ、ストライド幅）
  - get_Uncacheable(size)
    - uncacheable属性領域を確保し、先頭アドレスを返す
- これらはいずれも評価用であり、改良・検討の余地多

行列積の例(blocking, 4x4unroll有り)

on-chip memory量に応じてこの宣言のみを変更

オンチップメモリ領域を3分割
オンチップメモリサイズから
ブロックサイズを考える

```
#define BL 36
  OnChip = (double *)get_OnChipAddr();
  A = &OnChip[0];
  B = &OnChip[BL*BL];
  C = &OnChip[BL*BL*2];
  for (ii = 0; ii < N; ii+=BL){
    for (jj = 0; jj < N; jj+=BL){/*jje, kke, str_端数考慮*/
      p_load(&c(ii,jj), C, sizeC, jje*8, str_C);
      for (kk = 0; kk < N; kk+=BL){
        p_load(&a(ii,kk), A, sizeA, kke*8, str_A);
        p_load(&b(kk,jj), B, sizeB, jje*8, str_B);
        for (i = 0; i <= iie-4; i+=IU){
          for (j = 0; j <= jje-4; j+=JU){
            for (k = 0; k < kke; k++){
              /* A,B,Cを使って4 x 4 unroll 行列積, */   } }
              /*端の処理は省略*/}}
      p_store(C, &c(ii,jj), sizeC, jje*8, str_C);   }  }
```

まとめと現状

- SCIMAアーキテクチャの基本的な部分を紹介
  - 触れなかったところ:
  - page-load/store, load/store 等のメモリアクセス命令に関する依存関係の処理
- シミュレータを構築、種々のアプリケーションの評価中
  - 現状:オンチップメモリを陽に扱うプログラムをユーザが書いている
  - キャッシュ用最適化はパラメータチューニングが試行錯誤的(ブロックサイズなど)
  - SCIMA最適化は、オンチップメモリ使用の方針決定はかなり知的作業だが、そのあとは楽