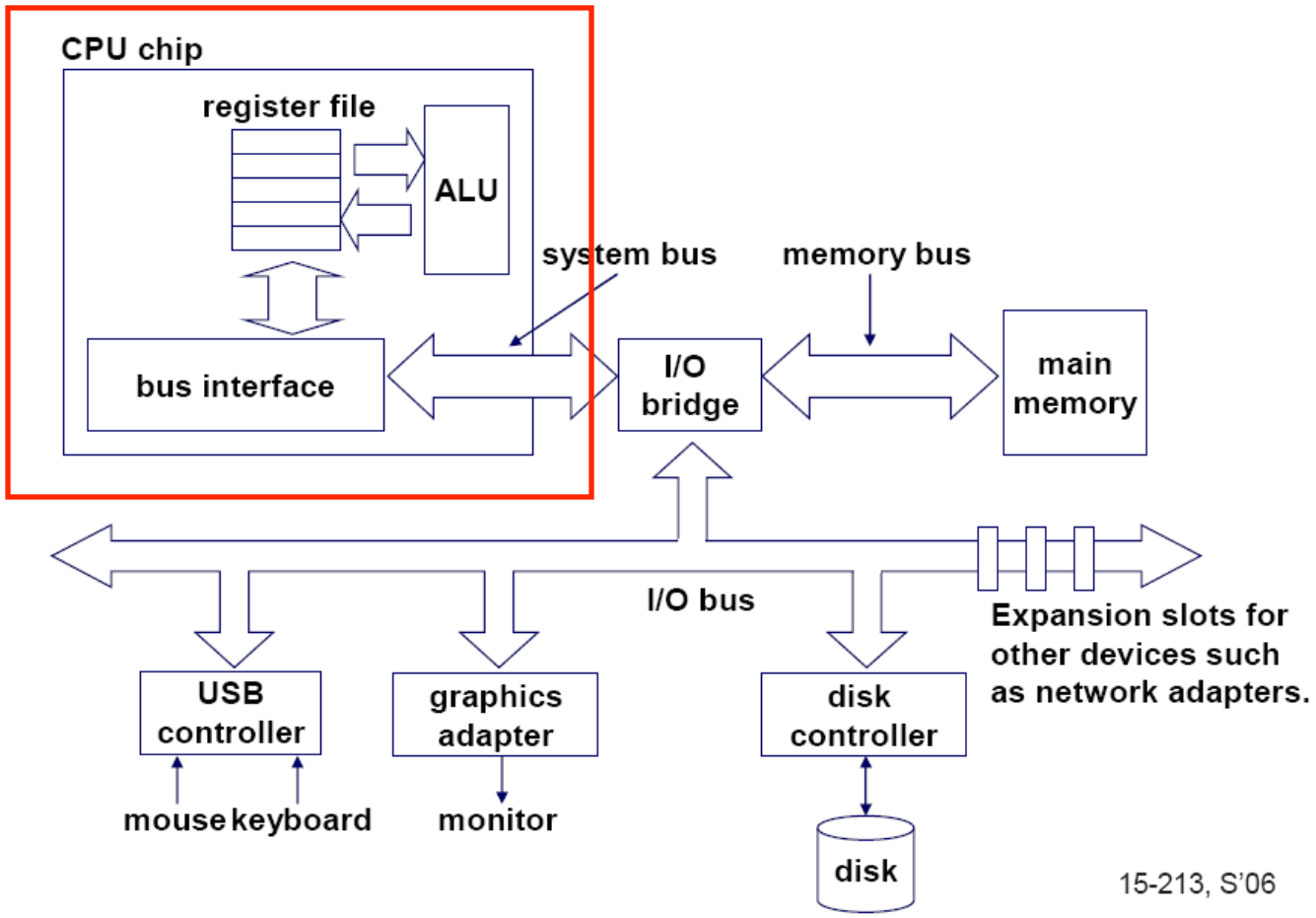
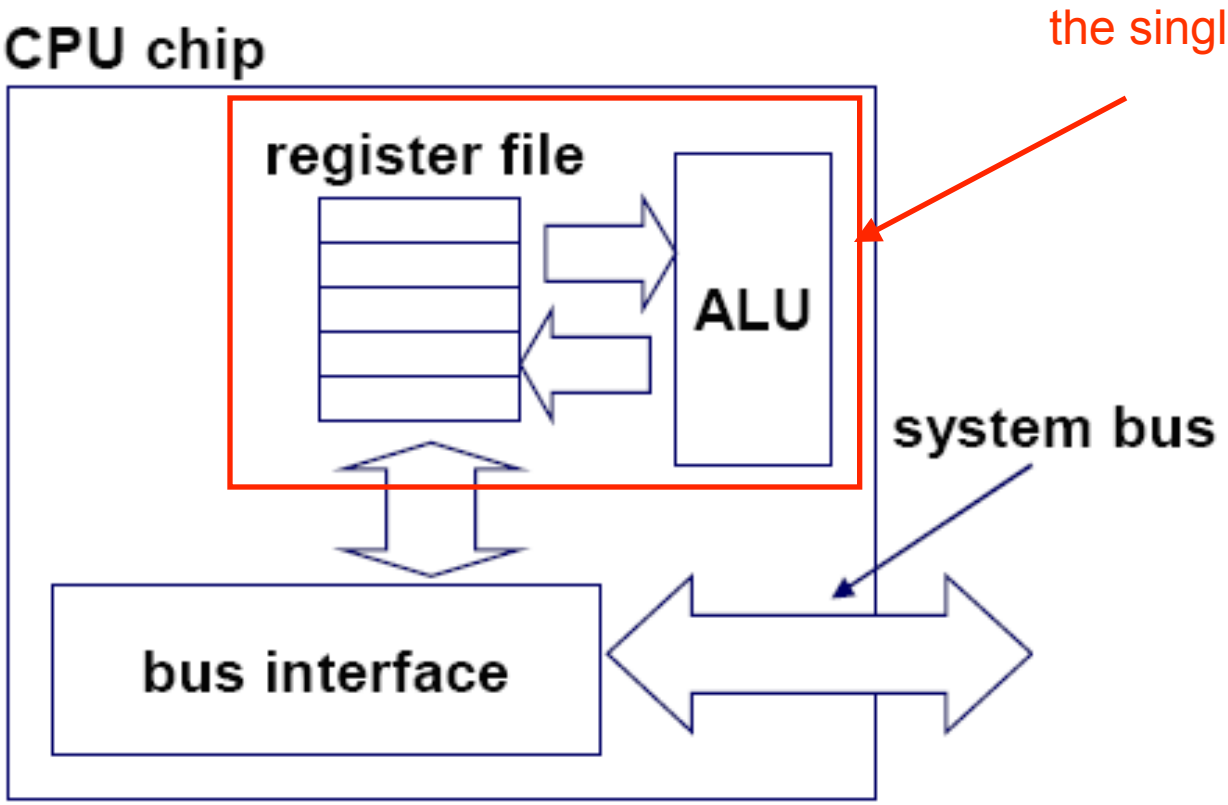
マルチコア・メニーコア アーキテクチャ (1)

シングルコアの計算機

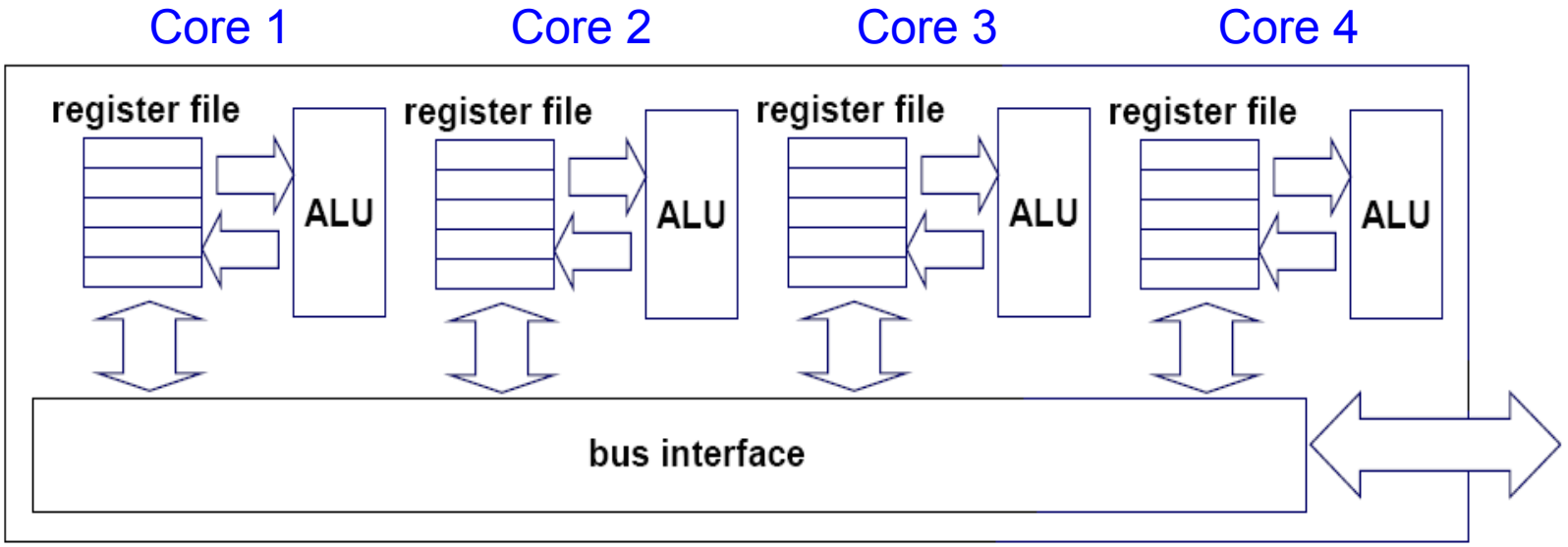

CPU chip
register file
ALU
system bus
memory bus
bus interface
I/O bridge
main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
graphics adapter
disk controller
mouse
keyboard
monitor
disk
15-213, S'06

CPUチップ


CPU chip
the single core
register file
ALU
system bus
bus interface

マルチコアアーキテクチャ

- マルチコアアーキテクチャとは、一つのCPUに、複数のコアが搭載されたもの
- コアのアーキテクチャは複雑なものから単純なものまで
  - 複雑なもの：x86 CPU, Cell BE etc.
  - 単純なもの：GPUなど



なぜマルチコアが必要か

- 手っ取り早い性能向上
- ひとつのコアのまま動作周波数を向上させることが困難になった
- アーキテクチャの複雑性
  - 設計と検証にかかるコストが膨大
  - マルチコアは「コピー」
  - 熱密度の問題
  - 困難点：並列性
- OSやアプリケーションはマルチスレッド化、並列化されているものもある

講義で紹介する様々なアーキテクチャ

- Cell B.E. by IBM/Sony/Toshiba
- x86 processor by Intel & AMD
- SPARC
- GPU by NVIDIA & AMD/ATI
- MIC by Intel
- PEZY-SC by PEZY

- 大型計算機の設計で発見発明されたことは、マイクロプロセッサで繰り返し利用されている
  - RISC, CISC
  - 並列処理
  - インターコネクト
  - SIMD, MIMD etc…

設計のポイント(1):命令制御方法

- マルチコアCPU
  - x86, Cell, SPARC
  - 複数のコアはそれぞれが独立して動作可能なプロセッサ
  - 実効的なベクトル長は4(AVX2) – 8(AVX512, HPC-ACE2)
  - 同じチップ(ダイ)の上に実装されているだけ
  - メモリ共有はキャッシュを介して、あるいは専用インターコネクト
- メニーコア type 1 : GPU
  - SIMDだが32 – 64コアごとに同一命令列
  - 各コア複数スレッドを実行するので、実効的なベクトル長は128など
- メニーコア type 2: GRAPE-DR
  - 完全なSIMD : 512コアが同一命令実行
  - ベクトル長は512 x 4 = 2048
  - 最も命令の利用効率がよい
- メニーコア type 3 : PEZY-SC
  - MIMD : 各コアは別の命令を実行可能だが、SIMD的に利用

設計のポイント(2):メモリ・Network on Chip(NoC)

- マルチコアCPU
  - 各コアはレジスタ、1次2次キャッシュをもつ
  - それより高次のキャッシュは複数コアで共有
  - NoC: リングやメッシュ、トーラスで接続
- メニーコア type 1 : GPU
  - 各コアがレジスタと1次キャッシュを持つ
  - レジスタが多い (SIMT)
  - 32,64コアごとに共有メモリあり
  - 専用のNoCはない
- メニーコア type 2: GRAPE-DR
  - レジスタとローカルメモリ
  - 共有メモリあり
  - Noc:総和ネットワーク

設計のポイント(3):演算器構成・命令セット(ISA)

- マルチコアCPU
  - 整数ALUと浮動小数点(FP)演算器
  - 既存ISAを踏襲するかしないか
    x86の場合OoOでかつレガシー命令もサポートされている
  - FP演算器はSIMD化されている
    - SSE2(2 DP ops), AVX2(4 DP ops), AVX512(8 DP ops)
- メニーコア type 1 : GPU
  - 整数ALUと浮動小数点(FP)演算器
  - ISAはそれ専用に開発されている
    - 通常はIn-orderのシンプルなもの
  - FP演算器はスカラー(1 DP ops)
  - NVIDIA:特殊関数ユニットは32コアで共有
- メニーコア type 2: GRAPE-DR
  - 整数ALUと浮動小数点(FP)演算器
  - FP addとFP mulのみ
  - 特殊関数ユニットなし

演算性能

- コア数 x コア当たりの演算数 x 動作周波数
  - 「コア」の定義は場合によるので注意が必要
  - FP性能の計算については、Fused-Multiply-Add演算器を考慮する
  - DP性能とSP性能の比は通常2:1だが、異なる場合もある

- x86 CPUの場合 (Haswell): CPU+GPU fused…
  - 2 (core) x 8(AVX2) x 2(FMA) x clock

特徴 [編集]

Ivy Bridgeから引き継ぐ特徴

- トライゲート22nmプロセス[9]
- 14段のパイプライン
- 1つのコアの余剰リソースを2つ目のコアに仕立てるハイパースレッディング・テクノロジー
- ターボブースト・テクノロジー

新規に確認されている特徴

CPU部

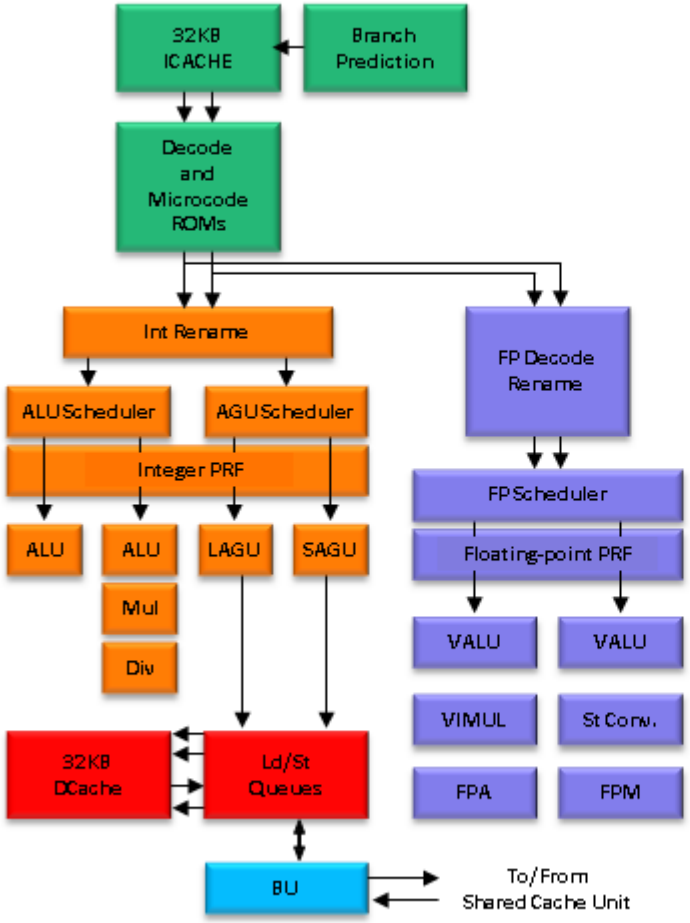
- 演算処理のためのポート数が6個から8個へ拡充[10]
- AVX2 (Advanced Vector Extension 2) のサポートによる数値演算処理性能の向上[11]
- 分岐予測の精度向上による、パイプラインストールの減少[10]
- FMA3(Fused Multiply Add)、BMI(Bit Manipulation Instruction Sets (英語版))などの拡張命令の追加[12]
- トランザクショナルメモリのハードウェアサポート
- L1データキャッシュの帯域倍増（ロード64Bytes/cycle、ストア32Bytes/cycle）
- L2キャッシュの帯域倍増(64Bytes/cycle)
- L2 TLB(Translation Lookaside Buffer)エントリ数の増加およびラージTLBのサポート[13]
- リオーダバッファのエントリ数の増加(168→192)
- 物理レジスタファイルの増加（整数160→168、浮動小数点144→168）
- リザベーションステーションのエントリ数の増加(54→60)

GPU部

例：AMD Family 16h Processor

Software Optimization Guide for AMD Family 16h Processors

http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/SOG_16h_52128_PUB_Rev1_1.pdf



Abbreviation	Descriptive Name
Int Rename	Integer Register Rename Unit
Floating-point PRF	Floating-point Physical Register File
Integer PRF	Integer Physical Register File
FP Decode Rename	Floating-point Decode / Rename Unit
ALU	Arithmetic and Logic Unit
LAGU	Load Address Generation Unit
SAGU	Store Address Generation Unit
BU	Bus Unit
VALU	Vector ALU
VIMUL	Vector Integer Multiply Unit
St Conv.	Store / Convert Unit
FPA	Floating-point Addition Unit
FPM	Floating-point Multiplication Unit
Ld/St Queues	Load / Store Queues

Diagram Legend

例：整数演算の部分のサマリー

Overall	out-of-order, 2 wide decode/dispatch/retire.			
	6 wide issue (2 alu, 1 ld, 1 st, 2 fp)			
	ISA: AMD64, x87, mmx, SSE1, SSE2, SSE3, SSSE3, SSE4a, SSE4.1, SSE4.2, AES/CLMUL, MOVBE, AVX, F16C, BMI1			
Integer				
	Integer Exec Units:		Pipes:	Pipe0: ALU (also handle FP->INT jam)
	o ALU (x2)			Pipe1: ALU, MUL, DIV
	o AGU (LAGU, SAGU)			Pipe2: LAGU
	o MUL, DIV			Pipe3: SAGU (also handles 3-operand LEA)
	Load latencies			
		normal integer load: 3 cycles		
		normal fp load: 5 cycles		
		misaligned load: +1 cycle		
		non-0 FS,GS: +1 cycle		
		load miss D$, hit L2$: +22 cycles		

例:整数演算のレイテンシ (16h: Jaguar)

Integer instructions

Instruction	Operands	Ops	Latency	Reciprocal throughput	Execution pipe	Notes
Arithmetic instructions						
ADD, SUB	r,r/i	1	1	0.5	I0/1	
ADD, SUB	r,m	1		1		
ADD, SUB	m,r	1	6	1		
ADC, SBB	r,r/i	1	1	1	I0/1	
ADC, SBB	r,m	1		1		
ADC, SBB	m,r/i	1	8			
CMP	r,r/i	1	1	0.5	I0/1	
CMP	r,m	1		1		
INC, DEC, NEG	r	1	1	0.5	I0/1	
INC, DEC, NEG	m	1	6	1		
MUL, IMUL	r8/m8	1	3	1	I0	
MUL, IMUL	r16/m16	3	3	3	I0	
MUL, IMUL	r32/m32	2	3	2	I0	
MUL, IMUL	r64/m64	2	6	5	I0	
IMUL	r16,r16/m16	1	3	1	I0	
IMUL	r32,r32/m32	1	3	1	I0	
IMUL	r64,r64/m64	1	6	4	I0	
IMUL	r16,(r16),i	2	4	1	I0	
IMUL	r32,(r32),i	1	3	1	I0	
IMUL	r64,(r64),i	1	6	4	I0	
DIV	r8/m8	1	11-14	11-14	I0	
DIV	r16/m16	2	12-19	12-19	I0	
DIV	r32/m32	2	12-27	12-27	I0	
DIV	r64/m64	2	12-43	12-43	I0	
IDIV	r8/m8	1	11-14	11-14	I0	
IDIV	r16/m16	2	12-19	12-19	I0	
IDIV	r32/m32	2	12-27	12-27	I0	
IDIV	r64/m64	2	12-43	12-43	I0	

例:浮動小数点演算のレイテンシ (16h: Jaguar)

Floating point XMM instructions

Instruction	Operands	Ops	Latency	Reciprocal throughput	Execution pipe	Notes
Arithmetic						
ADDSS/D SUBSS/D	x,x/m	1	3	1	FP0	
ADDPS/D SUBPS/D	x,x/m	1	3	1	FP0	
VADDPS/D VSUBPS/D	y,y/m	2	3	2	FP0	
ADDSUBPS/D	x,x/m	1	3	1	FP0	SSE3
VADDSUBPS/D	y,y/m	2	3	2	FP0	
HADD/SUBPS/D	x,x/m	1	4	1	FP0	SSE3
VHADD/SUBPS/D	y,y/m	2	4	2	FP0	
MULSS/PS	x,x/m	1	2	1	FP1	
VMULPS	y,y/m	2	2	2	FP1	
Math						
SQRTSS	x,x/m	1	16	16	FP1	
SQRTPS	x,x/m	2	21	21	FP1	
VSQRTPS	y,y/m	2	42	42	FP1	
SQRTSD	x,x/m	1	27	27	FP1	
SQRTPD	x,x/m	2	27	27	FP1	
VSQRTPD	y,y/m	2	54	54	FP1	
RSQRTSS/PS	x,x/m	1	2	1	FP1	
VRSQRTPS	y,y/m	2	2	2	FP1	

Intel Haswellアーキテクチャ

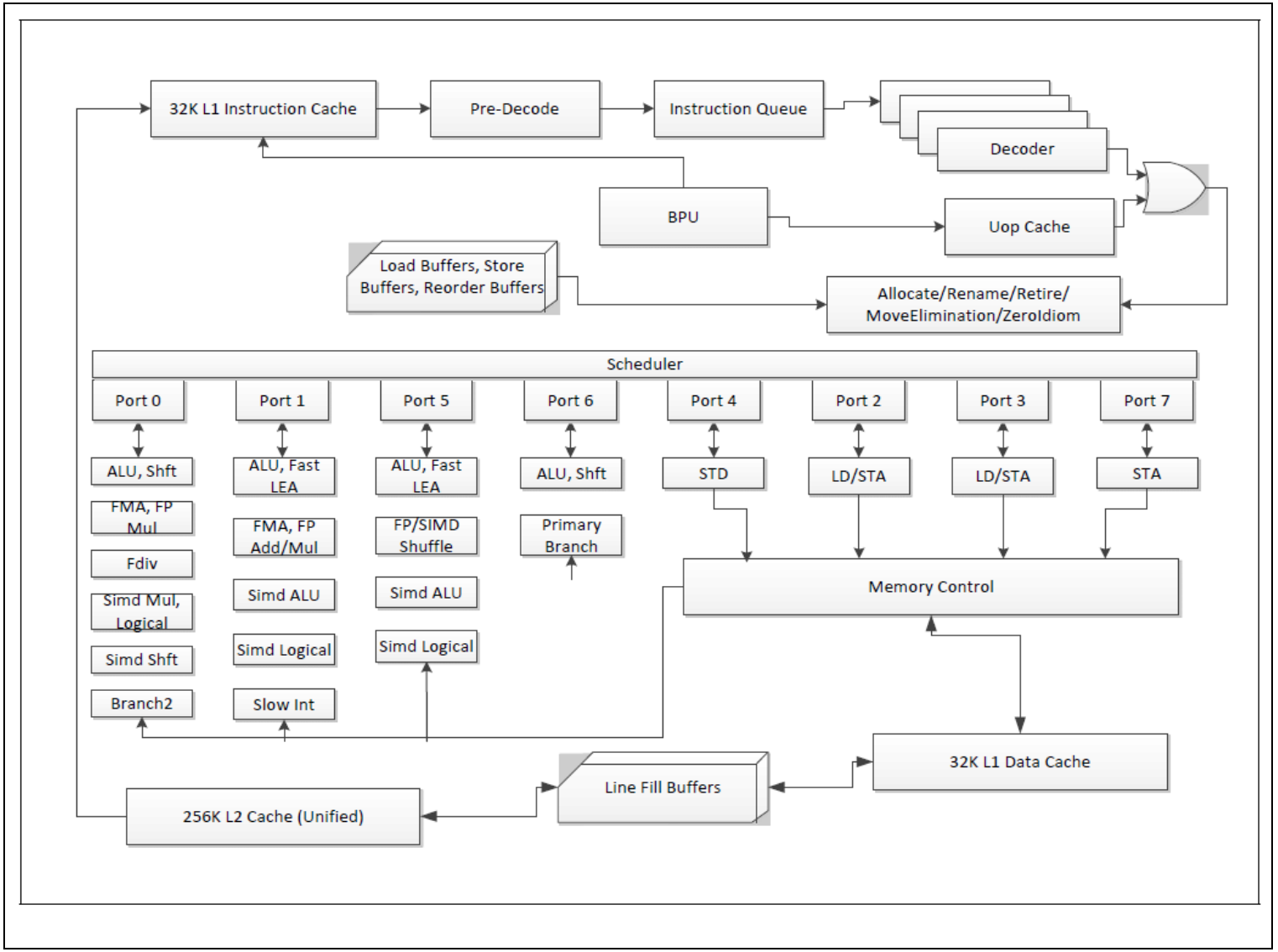
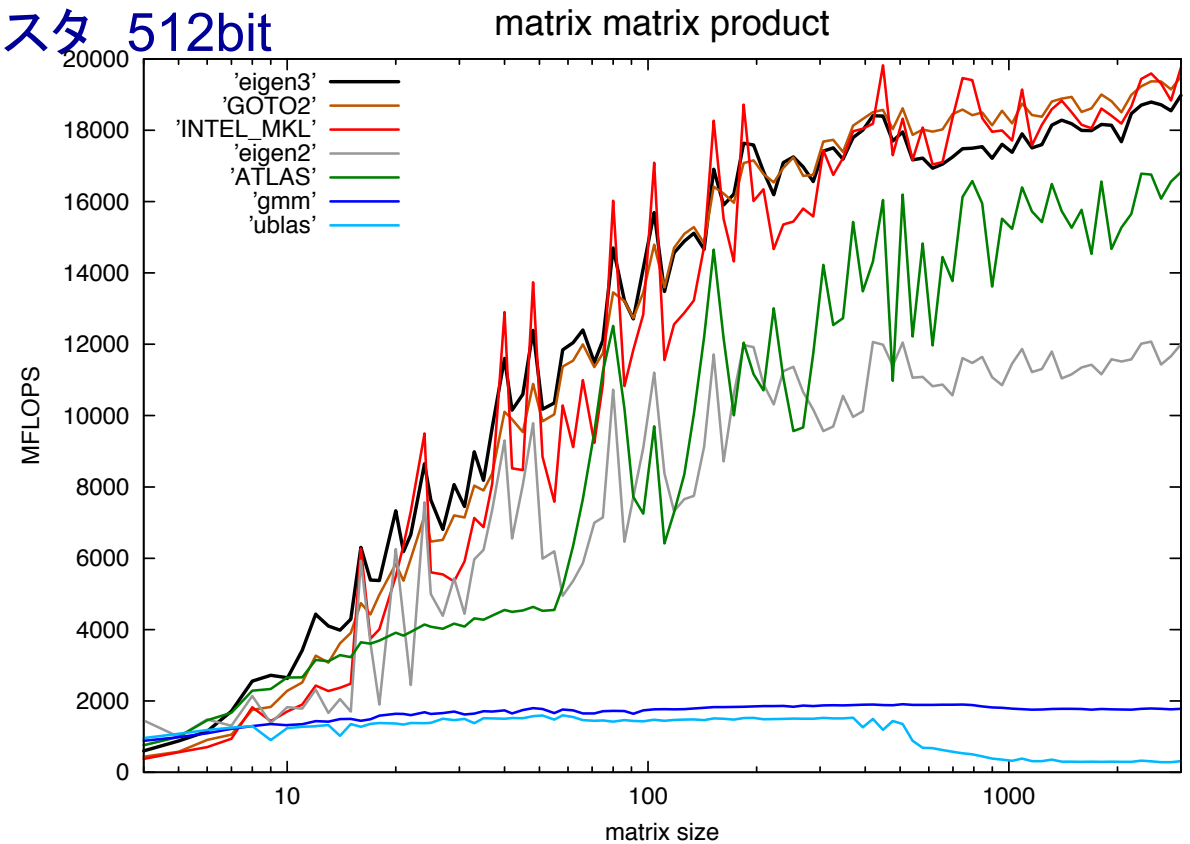


図 2-1 インテル® マイクロアーキテクチャー Haswell の CPU コア・パイプライン

SIMD命令による最適化

- 行列乗算の演習課題で、ブロック化をしただけでは、ピーク性能にほど遠い：SIMD命令を利用する必要あり
- 複数の演算を一度におこなうことができる
  - XMMレジスタ 128bit
  - YMMレジスタ 256bit
  - ZMMレジスタ 512bit



行列乗算の場合

- ブロックサイズ
  - 2x2, 2x4, 4x4 … 8x8
  - AVX2命令であれば倍精度4語なので4x4 or 4x8が最適のはず
  - b x bの行列乗算は：$2b^2$語読み出し, b^3演算
- 演算手法
  - 同時に計算できるパターンとは？(次ページ)
- キャッシュヒット
  - キャッシュライン64バイト(8語)

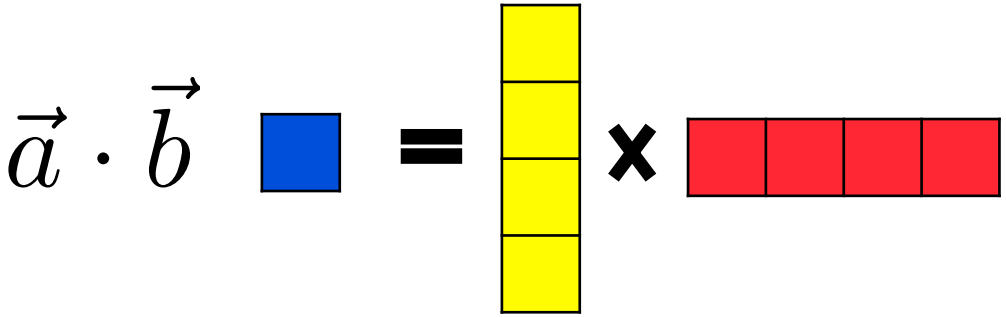
表 2-3 インテル® マイクロアーキテクチャー Haswell のキャッシュ・パラメーター

レベル	容量/アソシアティブ (ウェイ)	ラインサイズ (バイト)	最小レイテンシー[1]	スループット (クロック数)	ピーク帯域幅 (バイト/サイクル数)	アップデート方式
L1 データ	32KB/8	64	4 サイクル	0.5[2]	64 (ロード) + 32 (ストア)	ライトバック
命令	32KB/8	64	なし	なし	なし	なし
L2	256KB/8	64	11 サイクル	それぞれ異なる	64	ライトバック
L3 (共有)	それぞれ異なる	64		それぞれ異なる		ライトバック

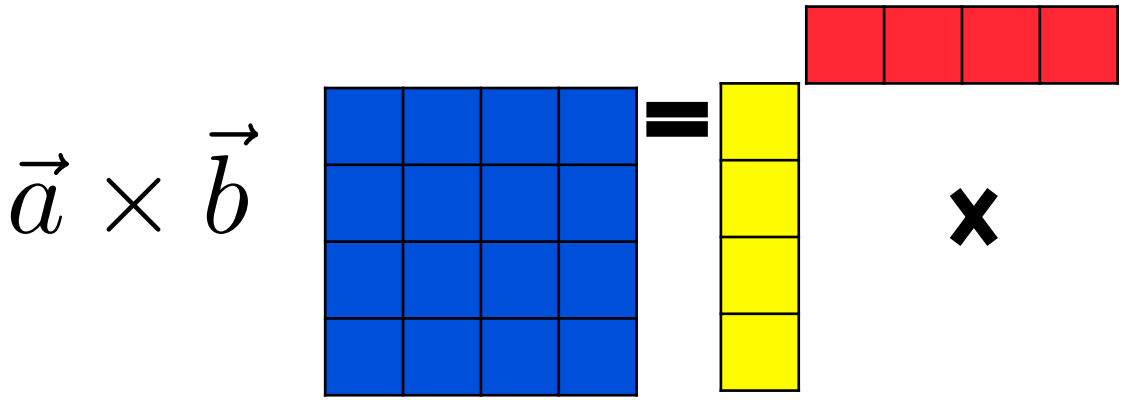
- ソフトウエアパイプライン

行列計算のSIMD化手法

- 内積型：スカラーアップデート
  - 行列Aから縦ベクトル、行列Bから横ベクトルを読み、内積を計算
  - 2ベクトルの読み込みあたり2b演算し、Cの1要素をアップデート



- 外積型：ベクトルアップデート SIMD向き
  - 行列Aから縦ベクトル、行列Bから横ベクトルを読み、外積を計算
  - 2ベクトルの読み込みあたり$2b^2$演算し、Cのb^2要素をアップデート



RDTSC : ReaD Time Stamp Counter

- Intel/AMDのCPUではCPUクロック数を計測することができる。以下のコード例を参考。

```
unsigned long long t1, t2;
RDTSC(t1);
// 時間計測したいコードをここに挿入する
RDTSC(t2);
printf("CPU Clock=%llu¥n", t2-t1);
```

```
#define RDTSC(X)¥
   asm volatile ("rdtsc; shlq $32,%%rdx; orq %%rdx,%%rax" : "=a" (X) :: "%rdx")
```

倍精度演算の場合の例： レイテンシは4クロック

コンパイルの方法	「a=a+a」を10回	「a=a*a」を10回
gcc -00	129	130
gcc -01（gcc -0と同じ）	50	51
gcc -02 / gcc -03	40	40
icc -00	130	145
icc -01	130	129
icc -02（icc -0と同じ）/ icc -03	40	40

http://www.cc.u-tokyo.ac.jp/support/press/news/VOL10/No5/200809tuning.pdf

最終課題について

- アルゴリズムを選択し、行列乗算のMPI並列化をおこなう
  - Broadcast-Broadcast-Compute
  - Broadcast-Compute-Roll
  - Compute-Roll-All
  - レポート2の単純な並列アルゴリズム

- 課題の条件:
  - ブロック化された行列ルーチンを使うこと
  - 行列サイズを変えて性能をMFLOPSで計測する
  - MPIのプロセス数を変化させる
  - aplisv1 or aplisv2で実行する
  - 最終日(2016年11月28日)にレポートの報告をおこなってもらう
  - 詳細については随時連絡する