

# x64 のスカラー, SIMD 演算性能を測ってみた

---

C++ MIX #10

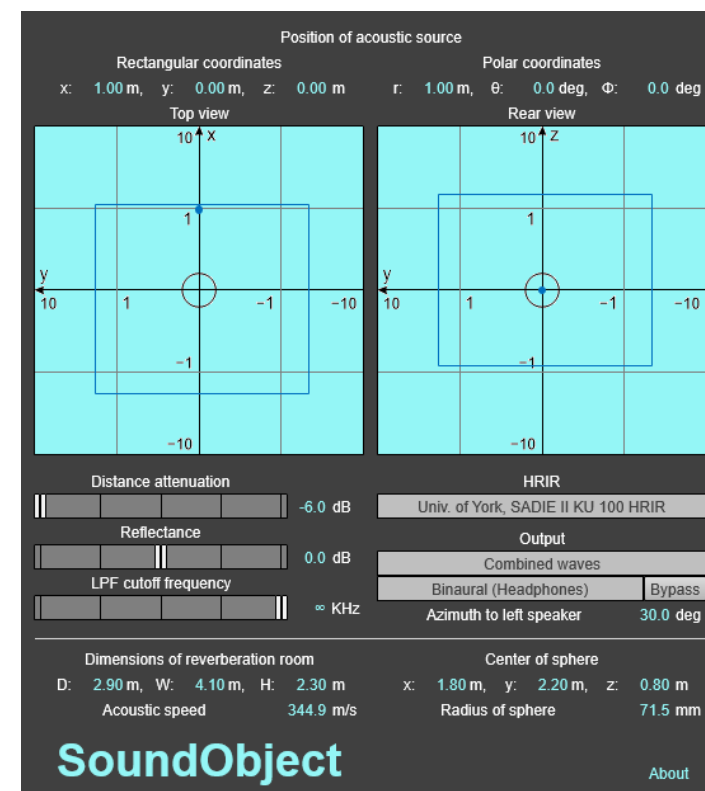
2024/4/19

 @suzumushi0

鈴木 宗良

# 背景

- DAW (Digital Audio Workstation: 楽曲作成ソフト) 向けのエフェクタープラグインを開発
  - 立体音響用プラグイン等のバイナリ, ソースを無償で公開
  - DAW はデジタル信号処理の実験プラットフォームとして最適
- プラグイン信号処理部の実体はリアルタイムの数値計算
  - 処理時間に上限
  - CPU 負荷は最大 1%-数% 程度
- 構想段階で, 実現可能性をどう判断する?
  - 作ってみなければ判らない, 出たところ勝負ではリスクが高い
  - 経験値から感覚論で判断?
  - 感覚論であっても CPU の演算性能に関する基本的なデータが必要



# 評価の目的と適用性

---

- 評価の目的

- Visual C++ による x64 スカラー, ベクトル数値演算命令のスループットを評価
  - 演算命令の直接記述やアセンブリ言語は使用しない

- 評価の適用性

- スループットはループ処理による繰り返し演算やベクトル演算の性能指標
  - 単発の演算ではレイテンシが重要な性能指標となるが, 今回は評価しない
- スループット性能は, 繰り返し回数やベクトルの次数による影響を強く受ける
  - コンパイラによって変換された機械語ルーチンの構造はこれらに依存
- Visual C++ の Vup, 他のコンパイラ, 他の CPU, GPU では全く異なる結果となる可能性は十分にある

# x64 数値演算の拡張命令

拡張命令	機能概要	VC++ オプション
SSE2	128 bit 長 16 本の XMM レジスタによる浮動小数点のスカラー, ベクトル演算 (SIMD), 整数のベクトル演算 32 bit 単精度浮動小数点の四則, 平方根, 逆数, 平方根の逆数演算命令 64 bit 倍精度浮動小数点の四則, 平方根演算命令 8, 16, 32, 64 bit 整数の加算, 減算命令, 16, 32 bit 整数の乗算命令	default
AVX2	256 bit 長 16 本の YMM レジスタによる浮動小数点のスカラー, ベクトル演算 (SIMD), 整数のベクトル演算 32 bit 単精度浮動小数点の四則, 平方根, 逆数, 平方根の逆数, 積和演算命令 64 bit 倍精度浮動小数点の四則, 平方根, 積和演算命令 8, 16, 32, 64 bit 整数の加算, 減算命令, 16, 32 bit 整数の乗算命令	/arch:AVX2
AVX-512	512 bit 長 32 本の ZMM レジスタによる浮動小数点のスカラー, ベクトル演算 (SIMD), 整数のベクトル演算 32 bit 単精度浮動小数点の四則, 平方根, 逆数, 平方根の逆数, 積和演算命令 64 bit 倍精度浮動小数点の四則, 平方根, 積和演算命令 8, 16, 32, 64 bit 整数の加算, 減算命令, 16, 32, 64 bit 整数の乗算命令	/arch:AVX512
AMX	512 bit 長ベクトル演算レジスタによる 8 bit 整数, 16 bit 半精度浮動小数点行列の積演算 行列の最大サイズは 1K Byte, 整数で 16 行 64 列, 浮動小数点で 16 行 32 列 演算結果は 32 bit 整数か 32 bit 単精度浮動小数点行列	未サポート

Note: Visual C++ オプションは最低要件を示す (機能制限では無い). デフォルトの SSE2 では, SSE4.1 や AVX-512 の命令 (pmulld, vpmullq) と, スカラー演算命令の双方に変換され, 実行時に cpuid に基づいて選択される.

# Visual C++ におけるベクトル演算

---

- Visual C++ コンパイラの自動ベクトル化機能
  - ループ処理における最適化機能の一つ
  - x64 ベクトル演算命令へ変換される場合がある
- C++ 標準ライブラリの `valarray`
  - 明示的にベクトル演算を記述
  - x64 ベクトル演算命令へ変換される場合がある
- Visual C++ 独自機能の組み込み (intrinsics) 関数
  - x64 ベクトル演算命令を直接記述
  - 今回は対象としない

# ループ処理の最適化

---

- 演算性能はコンパイラによる最適化に強く依存
- Visual C++ の最適化 /O2 は以下を含む
- ループ展開 (Loop unrolling)
  - for 文, while 文による演算のループを, スカラー演算命令を羅列したループに変換
  - Visual C++ でループ展開される条件は不明
- 自動ベクトル化 (Automatic vectorization)
  - for 文, while 文による演算のループを, ベクトル演算命令を羅列したループに変換
  - Visual C++ で自動ベクトル化される条件は不明確
  - 自動ベクトル化レポート出力オプション  
[構成プロパティ]>[C/C++]>[コマンドライン]>[追加のオプション] に /Qvec-report:2 を追加
  - 自動ベクトル化の抑止  
ループ記述の前に `#pragma loop (no_vector)` を挿入

# ループ展開の例

\$LL43@main:					
; 108 :	for (int j = 0; j < len; j++)				
; 109 :	*c32_p++ = *a32_p++ + *b32_p++;				
	mov	eax, DWORD PTR [r10]			
	lea	r10, QWORD PTR [rcx+20]		r10 update	
	add	eax, DWORD PTR [r9]		スカラー加算	
	lea	r9, QWORD PTR [rdx+20]		r9 update	
	mov	DWORD PTR [r11], eax			
	lea	r11, QWORD PTR [r8+20]		r11 update	
	mov	eax, DWORD PTR [rcx+4]			
	add	eax, DWORD PTR [rdx+4]		スカラー加算	
	mov	DWORD PTR [r8+4], eax			
	mov	eax, DWORD PTR [rdx+8]			
	add	eax, DWORD PTR [rcx+8]		スカラー加算	
	mov	DWORD PTR [r8+8], eax			
	mov	eax, DWORD PTR [rcx+12]			
	add	eax, DWORD PTR [rdx+12]		スカラー加算	
	mov	DWORD PTR [r8+12], eax			
	mov	eax, DWORD PTR [rdx+16]			
	mov	rdx, r9		rdx update	
	add	eax, DWORD PTR [rcx+16]		スカラー加算	
	mov	rcx, r10		rcx update	
	mov	DWORD PTR [r8+16], eax			
	mov	r8, r11		r8 update	
	sub	rbx, 1		loop カウンタ更新	
	jne	SHORT \$LL43@main			

- 5 回ループ展開された加算命令の例
- 2, 4, 5, 10 回のループ展開を確認

# エラーメッセージから推定した 自動ベクトル化の条件

対象は for 文, while 文によるループ (do 文は対象外)

```
for (int i; i < 1000; i++)
```

ループ変数はローカル変数のみ; ループ終了条件はループ開始時から不変; ループ変数更新は +1 のみ

{

NG: 依存関係のあるループ計算

NG: if, switch, break, continue 文等の条件分岐

OK: 浮動小数点の四則演算, sqrt (), 整数の加減乗算, math 関数 (演算速度は不変), inline 関数

NG: その他の関数

NG: bit 長が異なる変数への代入

NG: 32, 64bit 以外の構造体メンバへの参照

NG: 連続しない配列要素の参照

NG: 可変回数のシフト演算

} 等々謎

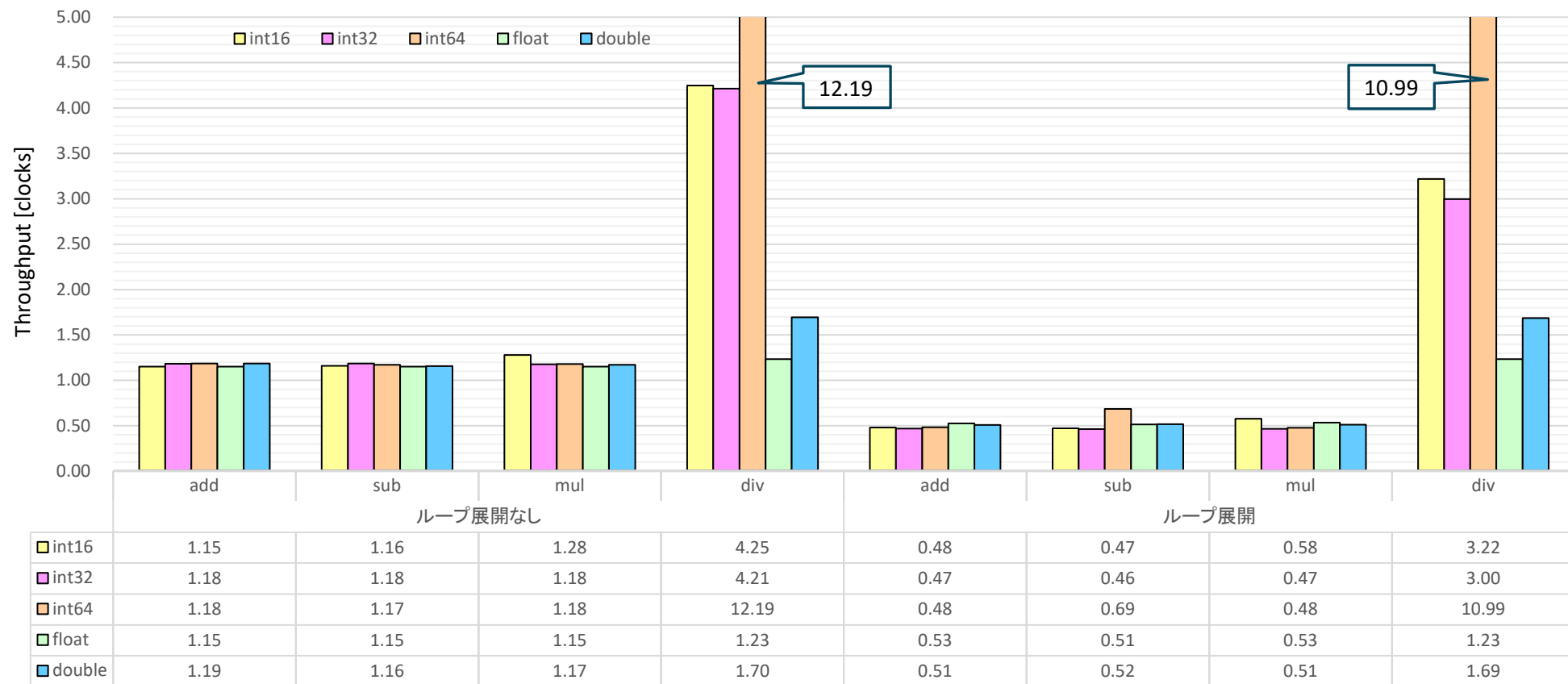


# 評価方法

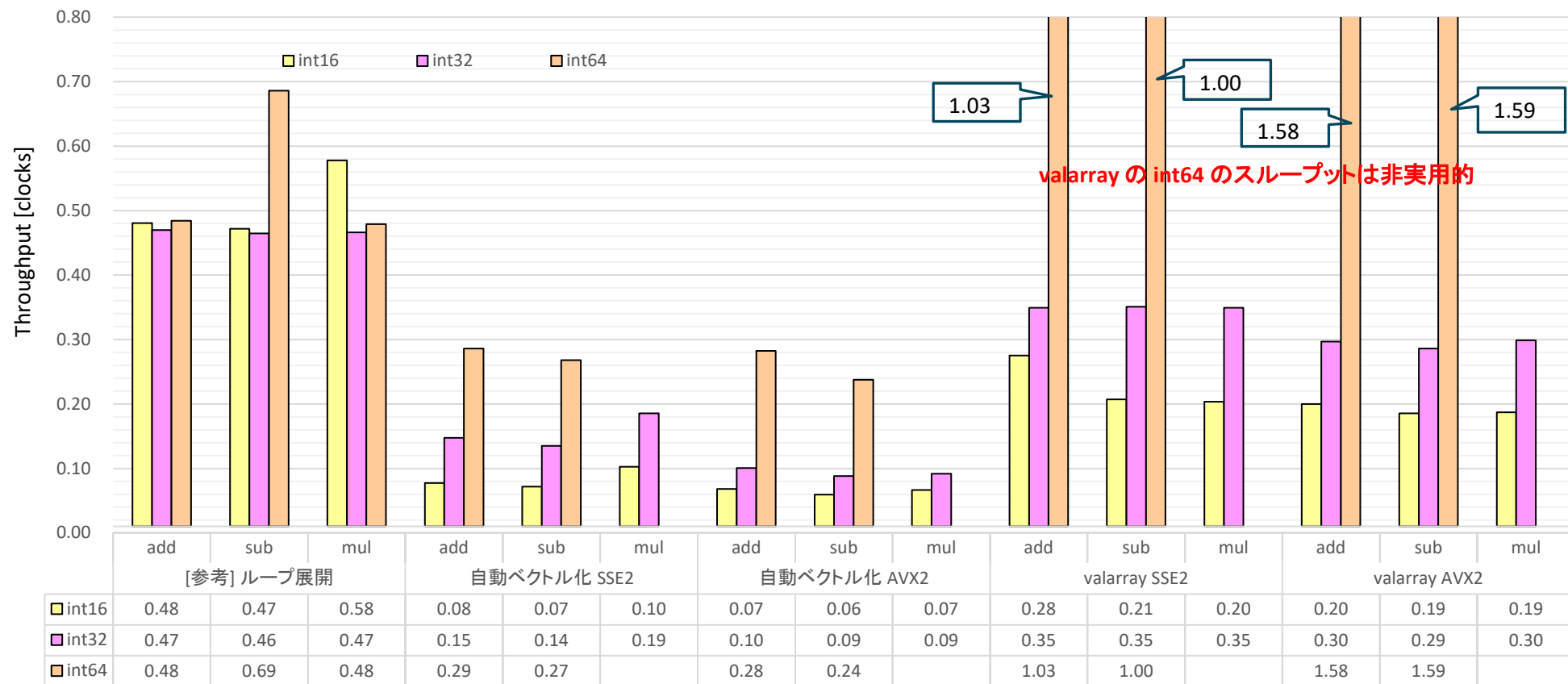
---

- 評価環境
  - Core i7 8565U 1.8GHz (8th Gen.)
  - Visual C++ v143
- 評価対象
  - int16 (short), int32 (int), int64 (long long), float, double の四則, 平方根演算
  - ループ展開無し, 有りにおけるスカラー演算
  - 自動ベクトル化, valarray によるベクトル演算 (SSE2, AVX2)
- 評価指標
  - 各演算を 1,000M 回繰り返す, 経過時間を計測
  - 1 回あたりの平均演算時間を CPU クロック数に換算し, スループットとする

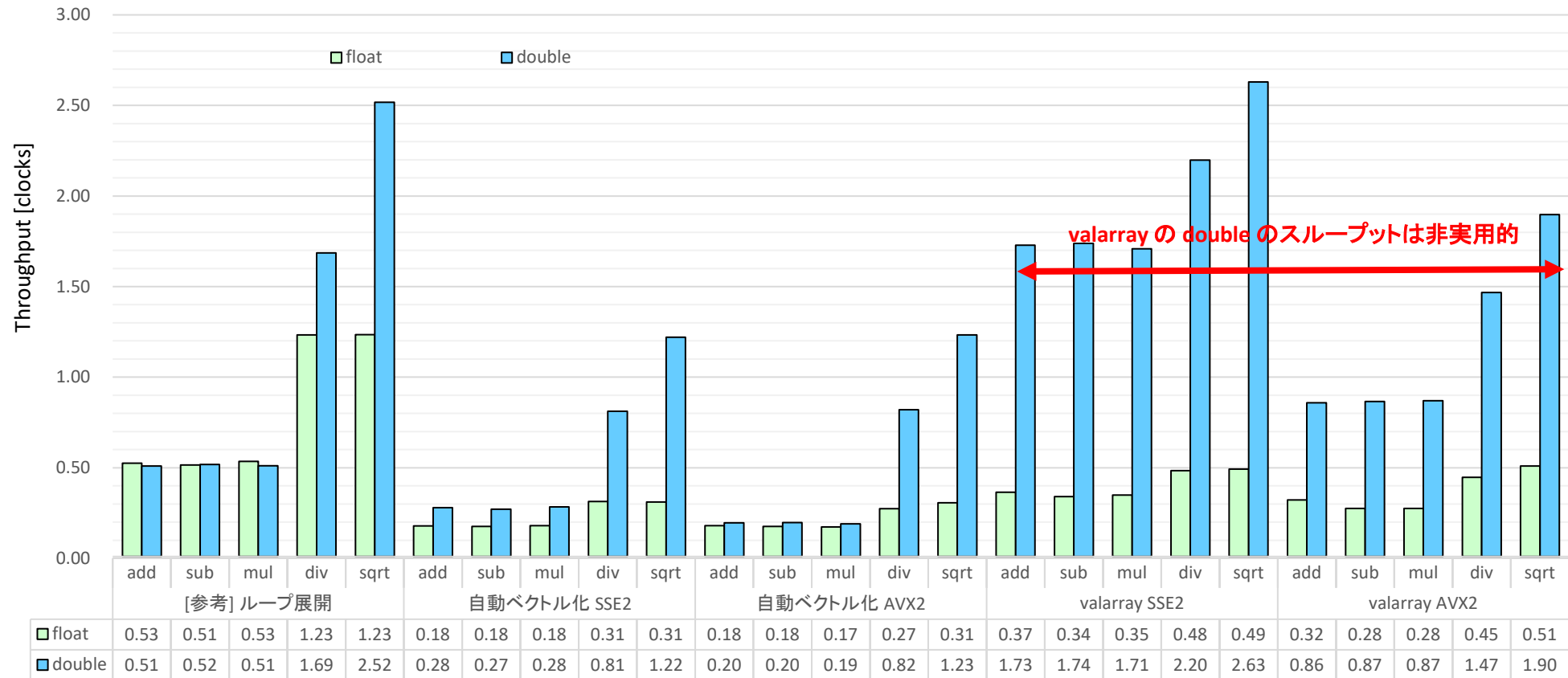
# スカラー演算のスループット



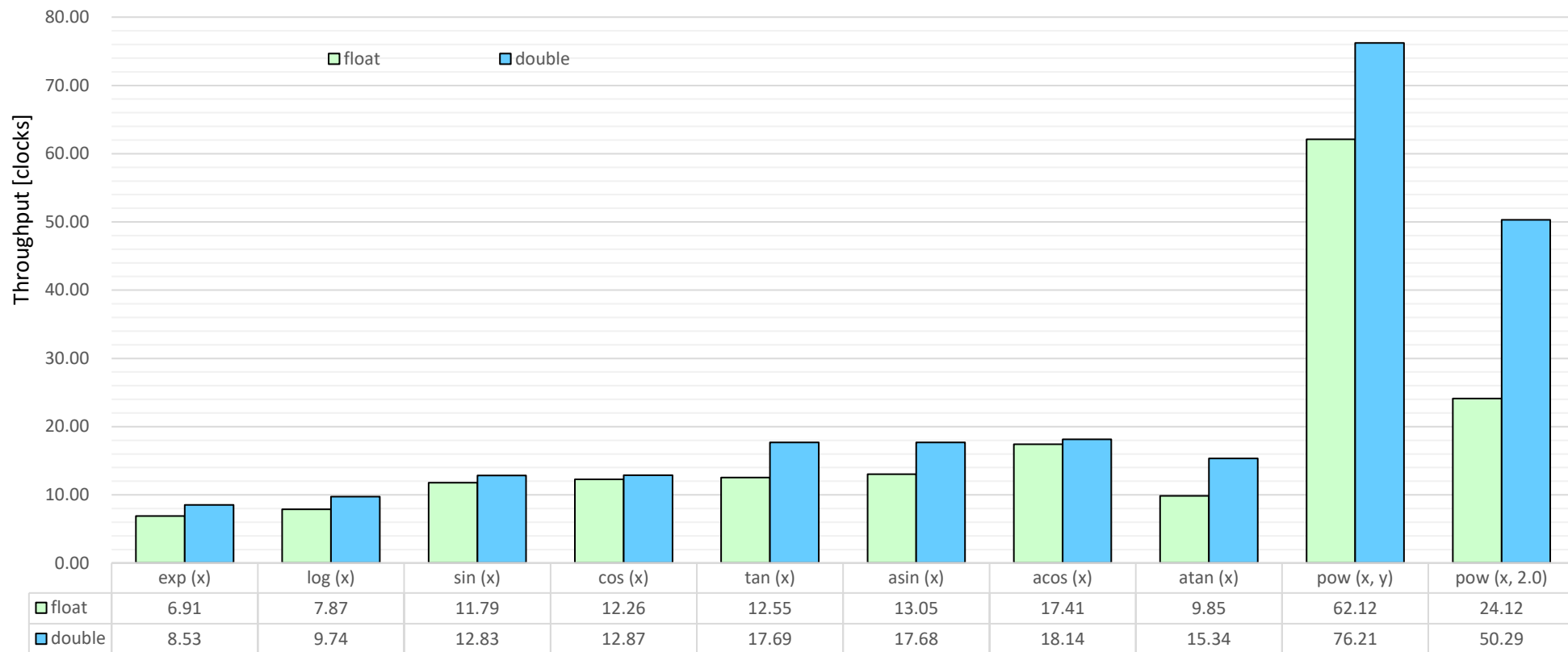
# 整数ベクトル演算のスループット



# 浮動小数点ベクトル演算のスループット



# [参考] 初等関数のスループット




# 結 果

---

- スカラー演算のスループット
  - 加減乗算のスループットは語長に係わらず整数, 浮動小数点で有意差は無い
  - 除算のスループットは, 浮動小数点の方が高い
- ループ展開と比較した, 整数ベクトル演算のスループット (自動ベクトル化)
  - SSE2 の int32 は, 加減算で約 3 倍, 乗算で約 2 倍に性能向上
  - AVX2 の int32 は 4 -5 倍に性能向上
- ループ展開と比較した, 浮動小数点ベクトル演算のスループット (自動ベクトル化)
  - SSE2 の float は, 加減乗算で約 3 倍, 除算平方根で約 4 倍に性能向上
  - SSE2 の double は約 2 倍の性能向上
  - AVX2 の double は, 加減乗算で約 3 倍の性能向上 (float と同等)
- その他
  - valarray は int64, double の演算性能に問題あり, また自動ベクトル化より性能が低い
  - int16 のスカラー乗算, 除算は 32bit 命令に変換されている
  - コンパイラは float の 逆数, 平方根の逆数演算命令, AVX2 の積和演算命令への変換はしない

# おわりに

---

- スループットをどう使う?
  - 例えば, サンプリングレートが 48KHz のデータを, クロック速度が 1.8GHz の CPU でリアルタイム処理
  - CPU 負荷を 1% 以下とすると 1 サンプル処理のクロック数の上限は  $1.8\text{GHz} / 48\text{KHz} / 100 = 375 \text{ clocks}$
  - 次数 67 の FIR フィルタは積和演算が 67 回となるから 67 clocks 程度必要か
  - 5 フィルタは行けるな, という感覚論
- x64 ベクトル演算は, 使える局面を選ぶが, 性能向上に確実に寄与する
  - Visual C++ はループ展開や自動ベクトル化の条件をドキュメント化して欲しい
- valarray はベクトル演算を簡潔に記述できる利点がある
  - Visual C++ は valarray の性能を, 自動ベクトル化と比較して遜色ないレベルにして欲しい
- 今回のスループット評価のソースコード 

# 参考資料

---

- Intel CPU 技術資料
  - Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4.
  - Intel® 64 and IA-32 Architectures Optimization Reference Manual Volume 1.
- x86, x84 CPU 命令セット一覧
- x86, x64 CPU 命令のレイテンシ, スループット一覧
- CPU の拡張機能サポート状況表示ツール Coreinfo