

Data Parallel C++ と OpenVINO で iGPU, NPU の計算速度とエネルギー 効率をを測ってみた

C++ MIX #14

2025/4/25



@suzumushi0

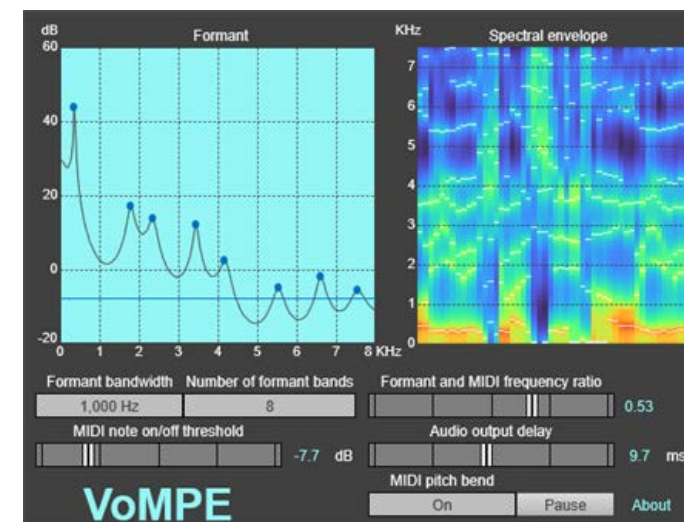
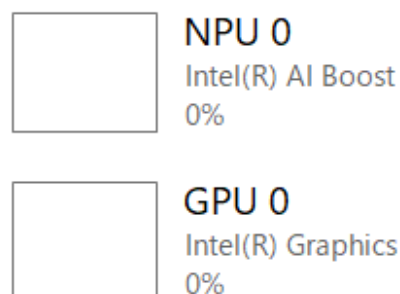


@suzumushi0.bsky.social

鈴木 宗良

自己紹介と背景

- 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, 音声情報処理プラグイン等のバイナリや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
 - 数値計算の高速化, リアルタイム処理に関心
- CPU 内蔵の iGPU や NPU を数値計算やオフロードに使えるのか?
 - 手元のパソコンで iGPU, NPU を性能評価してみた
 - eGPU と比較して iGPU は:
 - コア数が少なくクロック速度も遅い
 - 一方, パソコン買えば付いている
 - 評価観点は:
 - 計算速度
 - エネルギー効率



Core CPU (11 世代以降) 内蔵の iGPU Xe-LP (数値計算視点)

Xe-core			max 8 cores/GPU
	VXE (Xe vector engine, 旧 EU)		16 VXEs/core
		ALU (FP16 x16, FP32 x8)	256bit
		ALU (INT8 x32 INT16 x16, INT32 x8, FP64* x1, 数学関数 x2)	256bit
	L1 cache		max 192KB/core
L2 cache			max 4MB/GPU

- FP64 は Meteor Lake (Xe-LPG) 以降のサポート
- Arrow Lake 200H (Xe-LPG+), Lunar Lake (Xe2-LPG) では XMX による INT8, FP16 の行列計算をサポート
- 2 VXE をペア動作させるデュアルサブスライスをサポート (Xe2-LPG 以外)
- 公称性能 32 (256bit/INT8) x16 (VXE) x8 (core) x2 (積和) x2.35 (GHz) = 19TOPS
- 参考情報

Core Ultra CPU 内蔵の NPU

			NPU 3	NPU 4
NCE (Neural Compute Engine) tile			2 tiles/NPU	6 tiles/NPU
	DPU (Data Processing Unit)		1 DPU/tile	←
		MAC (Multiple-accumulate) unit (FP16 x2, INT8 x4)	512 units/DPU	←
	SHAVE DSP		2 DSPs/tile	←
		ALU (FP/BF32, FP/BF16, INT8, INT4)	128bit	512bit
	SRAM		2MB/tile	?

- Intel が 2016 年に買収した Movidius 社の LEON microcontrollers がベース
- Meteor Lake, Arrow Lake は NPU 3 (Movidius VPU3720), Lunar Lake は NPU 4 を搭載
- 公称性能 $4 (32\text{bit MAC/INT8}) \times 512 (\text{MAC}) \times 6 (\text{NCE}) \times 2 (\text{積和}) \times 1.95\text{GHz} = 48\text{TOPS}$
- 参考情報

Intel CPU 視点の数値計算向け CPU/GPU/NPU オフロード技術

		CPU	GPU (Intel)	GPU (NVIDIA, AMD)	NPU
ライブラリ利用	oneMKL	○	○		
	oneMath	○	○	○	
	OpenVINO	○	○		○
Data Parallel C++	DPC++ OpenMP	○	○	○	
	DPC++ SYCL	○	○	○	
ドライバ直叩き	OpenCL		○	○	
	oneAPI Level 0		○		○
	KMD Driver		WDDM	WDDM	MCDM

OpenMP とは

- 元々は, 共有メモリ型マルチプロセッサ向け並列処理 API
- HPC 分野では広く普及
- メリット: お手軽マルチスレッド化
- デメリット: 複雑な処理は厳しい
- OpenMP 4.0 以降は GPU 等のアクセラレータもサポート
- 参考情報

```
#include <iostream>
constexpr int N {4};

int main ()
{
    int a [N] {1, 2, 3, 4};
    int b [N] {5, 6, 7, 8};

    // GPU 処理開始
    #pragma omp target parallel for
    for (int i = 0; i < N; i++)
        b [i] *= a [i];
    // GPU 処理終了

    for (int i = 0; i < N; i++)
        std::cout << b [i] << std::endl;
}
```

SYCL とは

- C++ における CPU や GPU 等のアクセラレータによる並列処理 API
- テンプレートやラムダ式により, CPU 処理, GPU 処理をシームレスに記述
- GPU のメモリ構造を反映したバッファ-メモリモデル (CUDA と類似) と, 手軽に使える統合共有メモリモデルの 2 系統を規定
- [参考情報 1](#), [参考情報 2](#)

```
#include <sycl/sycl.hpp>
constexpr int N {4};
int main () {
    int a [N] {1, 2, 3, 4};
    int b [N] {5, 6, 7, 8};
    sycl::buffer<int> A (a, N);
    sycl::buffer<int> B (b, N);
    sycl::queue Q;
    // GPU 処理開始
    Q.submit ([&](sycl::handler& h) {
        sycl::accessor a (A, h, sycl::read_only);
        sycl::accessor b (B, h, sycl::read_write);
        h.parallel_for (N, [=](auto i) {
            b [i] *= a [i];
        });
    });
    // GPU 処理終了
    sycl::host_accessor ha (B, sycl::read_only);
}
```

OpenVINO とは

- 本来は ONNX, TensorFlow, PyTorch 等で学習したモデルを OpenVINO 中間形式に変換し, Intel CPU, GPU, NPU による推論を高速化する Python のツールキット
- 今回は OpenVINO の C++ API によりモデルを定義, OpenVINO 中間形式に変換, CPU, GPU, NPU で推論計算

// NxN 行列積演算のモデルを定義

```
auto A = std::make_shared<ov::op::v0::Parameter> (ov::element::f32, ov::Shape {N, N}); // 正方行列 A を定義
auto BT = std::make_shared<ov::op::v0::Parameter> (ov::element::f32, ov::Shape {N, N}); // 転置行列 BT を定義
auto matmul_A_BT = std::make_shared<ov::op::v0::MatMul> (A, BT, false, true); // 行列積演算を定義
auto result = std::make_shared<ov::op::v0::Result> (matmul_A_BT); // 計算結果を定義
auto model = std::make_shared<ov::Model> (ov::ResultVector {result}, ov::ParameterVector {A, BT}, "MatMul");

ov::Core ov_core; // OpenVINO runtime core を生成
ov::CompiledModel ov_ir = ov_core.compile_model (model, "GPU"); // モデルを中間形式に変換
ov::InferRequest infer_request = ov_ir.create_infer_request (); // inference request を生成

infer_request.set_tensor (A, ov::Tensor (ov::element::f32, ov::Shape {N, N}, a)); // 入力データを設定
infer_request.set_tensor (BT, ov::Tensor (ov::element::f32, ov::Shape {N, N}, bT)); // 同上
infer_request.infer (); // 推論 (計算) を実行
ov::Tensor output = infer_request.get_output_tensor (); // 推論 (計算) 結果を取得
const float* output_data = output.data<float> ();
```


評価方法, 評価環境

- 評価方法

- FP32 (CPU, GPU) と FP16 (GPU, NPU) による $N \times N$ 正方行列積の計算速度とエネルギー効率を測定
- OpenMP と SYCL では所謂タイル分割による行列積を実装, OpenVINO では行列積 API を利用
- OpenVINO による FP16 の行列積は float で記述, API で FP16 に変換, GPU, NPU の内部処理を FP16 とした
- 通常の正方行列積と, 転置行列による積の計算速度を比較し, 高速な方で評価
- GPU, NPU へのカーネル転送, データ転送, CPU のスレッド同期等のオーバヘッドを含めた時間で評価

- 評価環境

- Core Ultra 7 155U (L3 cache 12MB/socket)
 - GPU Xe-LPG 4 cores, 1.95GHz, 4TFLOPS (FP16), (L1 cache 64KB/core, L2 cache 4MB)
 - NPU 3, 700MHz, 2.9TFLOPS (FP16)
- Data Parallel C++ 2025.0.0, OpenVINO 2025.0.0
- GPU ドライバ 2.0.101.6557 (2025/1/28), NPU ドライバ 32.0.100.3714 (2025/2/3)

- CPU のモデル, コンパイラ, デバイスドライバのバージョンにより, 全く異なる結果となる可能性は十分にあります

- 評価には細心の注意を払っていますが, もし間違っていたらごめんね

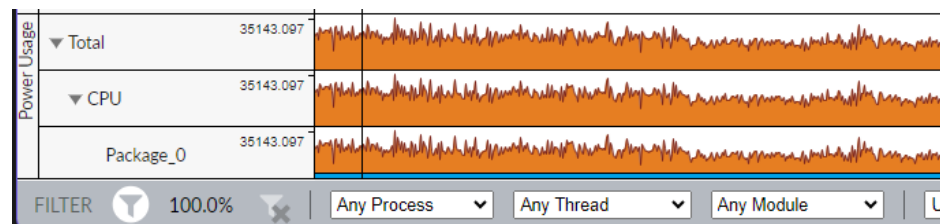
エネルギー効率の計測方法

- Intel VTune プロファイラでベンチマーク実行時の CPU パッケージのエネルギー消費量 (J) を測定

📄 Energy Consumption >

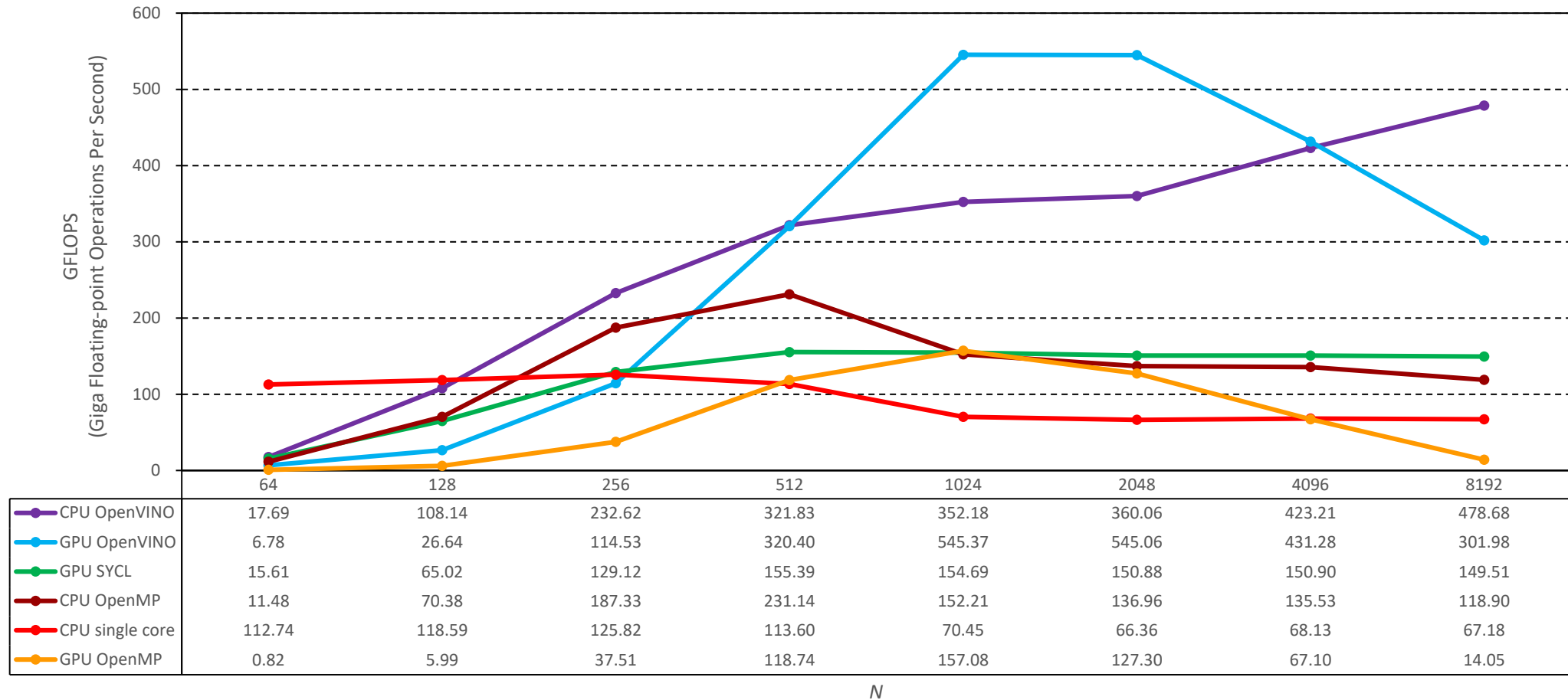
Profiled Entity Hierarchy	Energy Consumption (mJ)
Total	1,433,721.863
CPU	1,433,721.863
Package_0	1,433,721.863

**N/A is applied to non-summable metrics.*

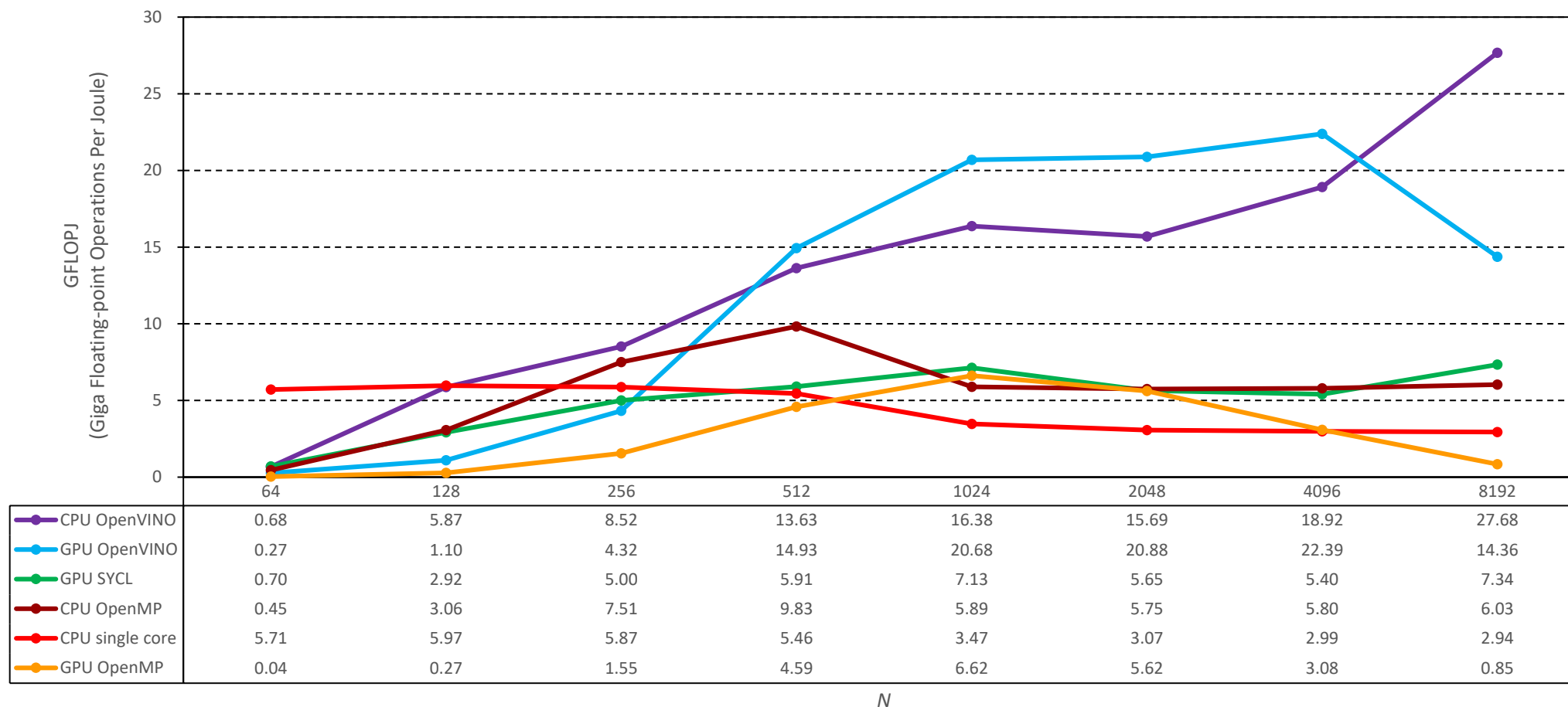


- 時系列データでエネルギー消費量がほぼ一定である事を確認
- エネルギー (J) あたりの計算回数 (Operations) をエネルギー効率 (Operations Per Joule: OPJ) とする
- Note:
 - 電力 (W) = 単位時間あたりのエネルギー (J /s)
 - よく見かける OPS /W という組み立て単位は (Operations /s) / (J /s) = (Operations Per Joule) であり時間とは無関係

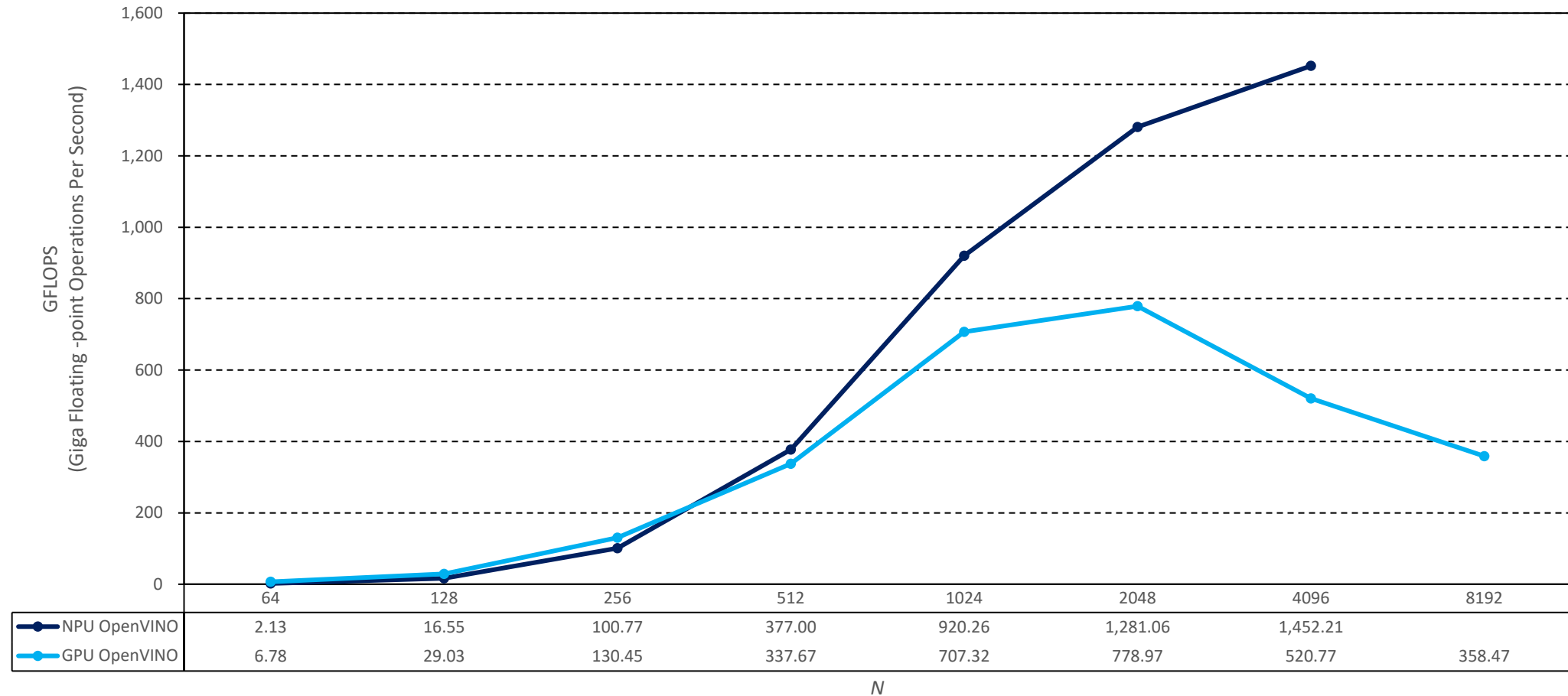
FP32 $N \times N$ 正方行列積の計算速度



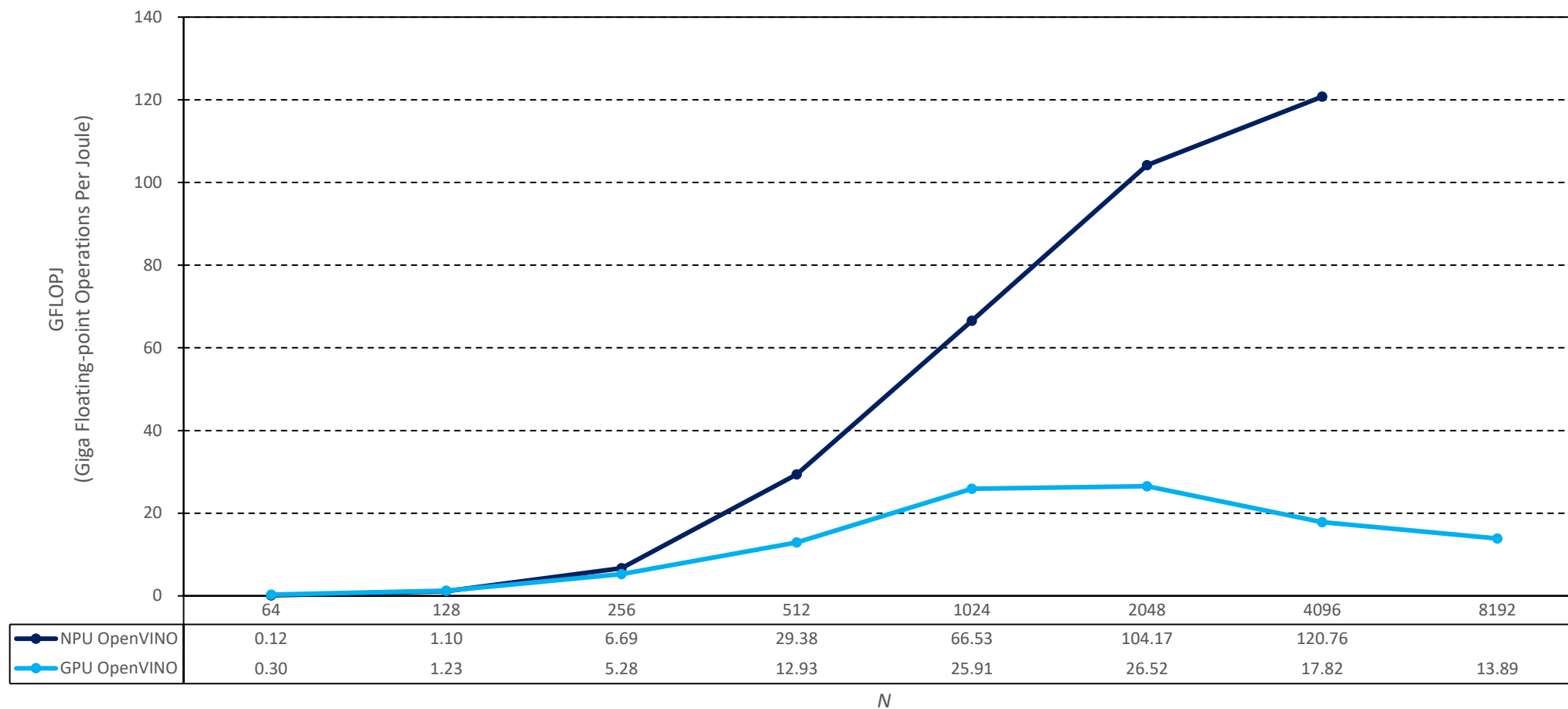
FP32 $N \times N$ 正方行列積計算のエネルギー効率



FP16 $N \times N$ 正方行列積の計算速度




FP16 $N \times N$ 正方行列積計算のエネルギー効率



結 果

- FP32 $N \times N$ 正方行列積の計算速度
 - 赤線は CPU シングルスレッド, $N = 512$ 以上で行列サイズの合計が L2 cache 2MB を超え, 性能低下
 - 茶線は OpenMP による CPU マルチスレッド (2P & 8E cores), シングルスレッドのほぼ倍の性能
- FP32 $N \times N$ 正方行列積計算のエネルギー効率
 - CPU の消費電力は, シングルスレッド約 20-23W, マルチスレッド 約 23-25W, OpenVINO 約 23-27W
 - GPU 処理の際の消費電力は OpenMP 約 22-25W, SYCL 約 22-28W, OpenVINO 約 22-27W
 - 何れも消費電力に大差は無いため, エネルギー効率と計算速度は同様の傾向
- FP16 $N \times N$ 正方行列積の計算速度
 - GPU OpenVINO の計算速度は FP32 より多少高速. 但し, 計算結果に数% の誤差があった (BF16 ?)
 - NPU OpenVINO の計算速度は極めて高速 (計算結果の誤差は 0.1% 程度)
- FP16 $N \times N$ 正方行列積計算のエネルギー効率
 - 消費電力は GPU OpenVINO 約 25-29W, NPU OpenVINO 12-15W
 - NPU のエネルギー効率は極めて高い

おわりに

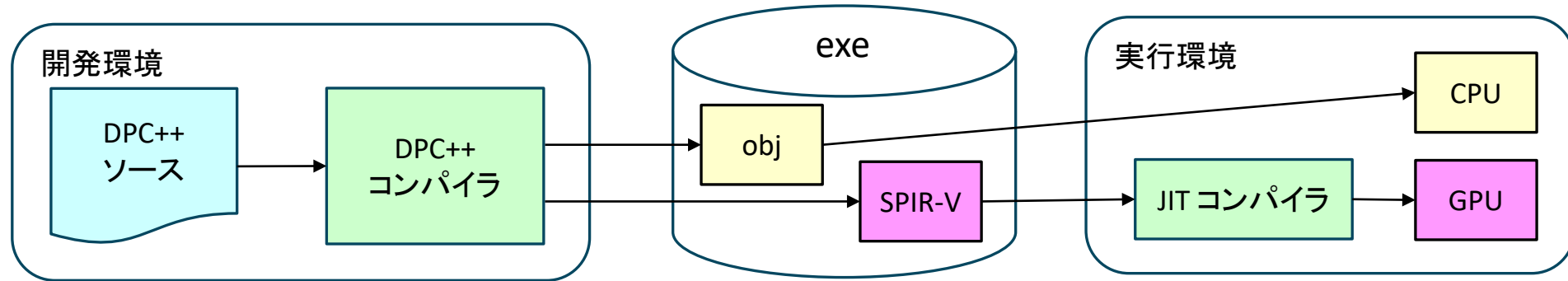
- まとめ
 - iGPU は CPU 処理のオフロード技術として有効
 - NPU は FP16 処理のオフロード, 高速化, 低消費電力化に有効
 - GPU, NPU 対応ライブラリは有用
- 多様な実行環境への対応が課題
 - AMD, ARM CPU, NVIDIA GPU 搭載 PC への対応
 - CPU モデル毎の iGPU, NPU の性能チューニング, 検証
 - GPU, NPU ドライバ更新の度に動作確認
-  ソースコードとコンパイル方法は公開していますので, ツッコミ歓迎
 - 今回使用したソフトは全て無料, ノート PC で動作

參考資料

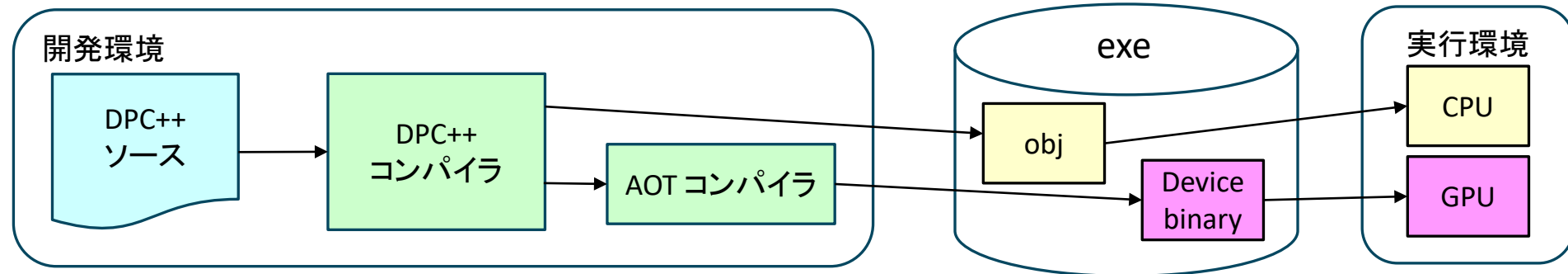
- Intel Data Parallel C++
 - [Get Started Guide](#)
 - [Intel® oneAPI Programming Guide](#)
 - [oneAPI DPC++ Compiler and Runtime architecture design](#)
 - [Developer Guide and Reference](#)
- Intel OpenVINO
 - [OpenVINO™ Runtime API Tutorial](#)
 - [Running and Integrating Inference Pipeline](#)
 - [Model Representation in OpenVINO™ Runtime](#)
 - [Interactive Tutorials \(Python\)](#)

参考: DPC++ における GPU カーネルの JIT, AOT コンパイル

- JIT (Just In Time) コンパイルの流れ (デフォルト)



- AOT (Ahead Of Time) コンパイルの流れ (Intel GPU が対象)



- 詳細なフローはこちらを参照