行列計算の実装とデバイスの比較

神宮司明良

概要

Day1, サーベイ

- 行列計算の応用先、高速化の研究や方法について調査
- 最終発表に向けたプレゼン資料を準備

Day2, 行列計算をプログラムして実装

- 典型的な行列ベクトル積をC++で実装し時間を計測
- ライブラリとの比較(Numpyなど)

Day3, 高速化について検討

- より速くするには？GPUやFPGAは？比較結果を表に

Day4, スパース行列計算について検討(option)

- 疎行列ー密ベクトル積の実装をCPU/GPU/FPGAで比較

C++の実装例

```
// Naive
// f(x) = A・x
void f(float A[1000][1000], float x[1000], float y[1000]) {
    for (int i = 0; i < 1000; ++i) {
        for (int j = 0; j < 1000; ++j) {
            y[i] += A[i][j] * x[j];
        }
    }
}
int main() {
    float A[1000][1000];
    float x[100][1000], y[100][1000];
    // 初期化
    for (int i = 0; i < 1000; ++i) {
        for (int j = 0; j < 1000; ++j) {
            A[i][j] = float(i + j);
        }
    }
    for (int i = 0; i < 100; ++i) {
        for (int j = 0; j < 1000; ++j) {
            x[i][j] = float(i - j);
            y[i][j] = 0;
        }
    }
    clock_t start = clock();
    for (int i = 0; i < 100; ++i) { // DO N=100 times and measure time
        f(A, x[i], y[i]);
    }
    clock_t end = clock();
    const double time = static_cast<double>(end - start) / CLOCKS_PER_SEC * 1000.0;
    printf("time %lf[ms]\n", time);
    return 0;
}
```

最終的な実験結果の表のイメージ

- 実験結果が見やすいように表にまとめる
- 強調したい結果は太字にする
- 下記のフォーマットは参考までに

			密行列 1000×1000		95%疎行列 1000×1000		
	Platform	**Clock**	**Latency**	**Power**	**Latency**	**Power**	コメント
Base1	RP4	2.4 GHz	56.7 ms	5.0 W	5.7 ms	5.0 W	C++による3重ループの単純な実装
Base2	RP4	2.4 GHz	56.7 ms	5.0 W	5.7 ms	5.0 W	Base1 + 最適化オプションの追加
Base3	RP4	2.4 GHz	56.7 ms	5.0 W	5.7 ms	3.0 W	Base2 + OpenMPによる並列化
Numpy	RP4	2.4 GHz	18.9 ms	5.0 W	1.8 ms	5.0 W	Numpyを用いた実装
GPU	Jetson Nano	1.4 GHz	**10.0 ms**	5.0 W	1.8 ms	5.0 W	GPUでCupyを用いた実装
FPGA	Ultra96	0.3 MHz	18.9 ms	**2.0 W**	**1.5 ms**	**2.0 W**	FPGAでの実装