プロセッサアーキテクチャ特論レポート

1518511 川瀬　拓実

2018 年 11 月 8 日

1　課題 1

iPhone7
製造会社 Apple
型番 Apple A10
アーキテクチャ ARM
FLOPS は不明、おそらく 300 GFLOPS 程度だろうと推測されている。
HUAWEI 5.2 型 P10 lite
製造会社 HiSilicon
型番 Kirin658
アーキテクチャ ARM
FLOPS は 40.8 GFLOPS

2　課題 2

動作周波数が 3.0GHz で、1 クロックあたり加算と乗算を 8 回同時に行えるので、この CPU の理論性能 FLOPS は $\mathrm{FLOPS} = 8 \times 2 \times 3.0\mathrm{G} = 48\mathrm{G(FLOPS)}$ となる。

0.002 秒あたり 10000000 回の加算と乗算を行なっているので、1 秒あたりに換算すると $\mathrm{FLOPS} = 2 \times 10000000 \div 0.002 = 10\mathrm{G(FLOPS)}$ となる。

理論性能との比から、使用効率は $10 \div 48 \simeq 0.208$ となる。

3　課題 3

定数の計算部分 pi * pitch/lambda は結果が同じになるため、ループごとに計算するのは効率の観点から望ましくない、ループの外で計算して保持する方が良い。また、徐算が行われているので予め小数に直して掛け算にすることで、処理に必要なサイクルが減り高速化が望める。配列のラベルを、配列を使うときに計算するのではなくループの最初の部分、すなわち他の浮動小数点演算をするときに行うことで、整数演算ユニットと浮動小数点数演算ユニットが独立に計算できるので、スーパースカラ実行ができ高速化が見込める。例として、配列を使用するときに img_d[i * N_x + j] のようにするのではなく、コード上の不動小数点演算をする部分付近にて、新しい変数を使用して label = i * N_x + j のように配列のラベルを予め用意しておくことで効率化できる。また、このプログラムはループが多いため、ループアンロールを行い条件分岐の回数を減らすことも効率化に寄与すると考えられる。

4　課題 4

パイプラインの処理時間は以下の式で表される。

$$(I + D - 1) \times P$$

I:実行する命令数

D:パイプラインのステップ数

P:パイプラインピッチである。これらをとに適応すると、それぞれの処理時間はパイプラインフラッシュを考慮しないと

:$(3 + 5 - 1) \times 12 = 84$:$(5 + 5 - 1) \times 8 = 72$

となる。単位はいずれもナノ秒である。ただし、実際の状況ではパイプラインフラッシュが起こり、その確率は段数が多いほど高まる。1 段の 1 ナノ秒あたりのパイプラインフラッシュの起こる確率を P とすると、1 マイクロ秒あたりおよそ 111 回のパイプラインフラッシュが起こると両者の処理時間がおおよそ等しくなる。

5　課題 5

アセンブラを上から読んでいくと、main 関数の中で各 int 型の値をレジスタにコピーしていき、jmp のところでラベル L2 に跳んでいる。ラベル L2 の中では、まず i の中身を比較、つまり $24(\%esp) - 2$ が 0 以下になるかどうかで条件分岐を行い、0 以下ならばラベル L3 に跳ぶことになっている。ここで for ループの処理が行われ、i が 0 から 2 であれば、条件分岐としてラベル L2 の中で c に対して値を足していく処理が行われる。for loop の処理はラベル L2 内の cmpl 命令と jle 命令とラベル L3 内で全て行われているため、L3 の前で loop 処理をしたいのであれば、cmpl 命令と jle 命令をラベル L3 の前の更に jmp 命令の前に移せば良い。

6　課題 6

6.1　ラベル L3 の中

LDR 命令により fp の値を r2 に読み込む。このとき#-8 がオフセットとして与えられる。初期値としてレジスタ r2 には 0 が入る。

MVN 命令により即値 15 にビット毎の論理 NOT 演算を施したものをレジスタ r3 に入れる。

MOV 命令により r2 レジスタに、r2 レジスタの中身を即値 2 ビットだけ算術右シフトした値をコピーする。

SUB 命令により r0 レジスタに、fp の値から 4 引いた値を代入する。

ADD 命令により r2 レジスタに、r0 レジスタの値と r2 レジスタの値を足したものを代入する。

ADD 命令により r3 レジスタに、r2 レジスタの値と r3 レジスタの値を足したものを代入する。

LDR 命令により r2 レジスタに、r3 のアドレスの値を代入する。

LDR 命令により r1 レジスタに、fp アドレスの値を代入する。

MVN 命令により即値 27 にビット毎の論理 NOT 演算を施したものを r3 レジスタに代入する。

MOV 命令により r1 レジスタに、r1 レジスタの中身を即値 2 ビットだけ算術右シフトした値をコピーする。

SUB 命令により r0 レジスタに、fp アドレスの値から 4 引いた値を代入する。

ADD 命令により r1 レジスタに、r0 レジスタの値と r1 レジスタの値を足したものを代入する。

ADD 命令により r3 レジスタに、r1 レジスタの値と r3 レジスタの値を足したものを代入する。
LDR 命令により r3 レジスタに、r3 のアドレスの値を代入する。
ADD 命令により r2 レジスタに、r2 レジスタの値と r3 レジスタの値を足したものを代入する。
LDR 命令により r1 レジスタに、fp アドレスの値を代入する。
MVN 命令により即値 39 にビット毎の論理 NOT 演算を施したものを r3 レジスタに代入する。
MOV 命令により r1 レジスタに、r1 レジスタの中身を即値 2 ビットだけ算術右シフトした値をコピーする。
SUB 命令により r0 レジスタに、fp アドレスの値から 4 引いた値を代入する。
ADD 命令により r1 レジスタに、r0 レジスタの値と r1 レジスタの値を足したものを代入する。
ADD 命令により r3 レジスタに、r1 レジスタの値と r3 レジスタの値を足したものを代入する。
STR 命令により r2 レジスタの値を r3 のアドレスに書き込む。
LDR 命令により r3 レジスタに、fp アドレスの値を代入する。
ADD 命令により r3 レジスタに、r3 レジスタの値と即値 1 を足したものを代入する。
STR 命令により r3 レジスタの値を fp アドレスに書き込む。

6.2　ラベル L2 の中

LDR 命令により r3 レジスタに、fp アドレスの値を代入する。
CMP 命令により r3 レジスタの値と即値 2 を比較する。
BLE 命令により上記の CMP の結果が、”より小さいか等しい”であれば分岐、L3 ラベルに移動する。
LDR 命令により r0 レジスタに、ラベル L4 から得た値を代入する。
LDR 命令により r1 レジスタに、fp にオフセット-44 を加えたアドレスの値を代入する。
LDR 命令により r2 レジスタに、fp にオフセット-40 を加えたアドレスの値を代入する。
LDR 命令により r3 レジスタに、fp にオフセット-36 を加えたアドレスの値を代入する。
BL 命令によりサブルーチン printf を呼び出す。
MOV 命令により r3 レジスタに、即値 0 を代入する。
MOV 命令により r0 レジスタに、r3 レジスタの値を代入する。
SUB 命令により sp レジスタに、fp から即値 4 を引いた値を代入する。
LDMFD 命令により sp レジスタに、fp と pc の値を代入する。

7　課題 7

深層学習の分野での学習のプロセスはもちろんのこと、それまで人間が行なってきた各層の構成や学習率などのいわゆるハイパーパラメータを決定する、より包括的で大規模な計算の実行。現在スーパーコンピュータを用いて行われるシミュレーション、新しい物質（化合物や製薬など）を作るための計算や、天気予報などの地球規模や宇宙規模の計算の分野。量子コンピュータの発明に向けて、古典的コンピュータによる量子コンピュータのシミュレーション。これらのもはや人間の手には追えない自由度の高すぎる系でのコンピュータによる計算はこれから発展が期待できる。特に量子コンピュータについては従来のコンピュータ（古典的コンピュータ）が苦手とする並列計算が容易にできることから新たな数値計算の分野が多数生まれることは想像に難くない。