整数定数による除算のための最良の命令列生成法

藤波 順久
株式会社ソニー・コンピュータエンタテインメント

概要

整数定数による除算を乗算を用いて高速に行う方法は、Granlund らが 1994 年に提案し、GCC にも実装されている。この方法を符号なし除算に適用した場合、除数によってはオペランドがワード長を超えるため、長い命令列を必要とする。一方、Fog は 1996 年に、これと似ているが一つではなく二つの基本式を用いることで、ワード長を超える演算を避ける方法を提案しているが、適用条件が不明確であった。発表者は、後者の方法の適用条件を厳密に求めることで、最良の命令列を生成できるようにした。この命令列は条件分岐を含むが、分岐確率が偏っているため、分岐予測を行うプロセッサでは良好な性能を期待できる。

Generating Optimal Instruction Sequences for Integer Division by Constants

FUJINAMI Nobuhisa
Sony Computer Entertainment Inc.

Abstract

Granlund et al. proposed in 1994 a method for integer division by constants using multiplications. It is implemented in GCC. When it is used for unsigned division, some divisors require operands that exceeds the word size, and long instruction sequences are required. On the other hand, Fog proposed in 1996 a similar method that avoids large operands. Using two instead of one basic expressions solved the problem. The applicability of the expressions were, however, not clear. The author sought for strict applicability conditions for the expressions in the latter method and enabled generating optimal instruction sequences for given divisors. The sequences may have conditional branches, but the probability is so skewed that satisfactory performance is expected on processors with branch prediction.

1 はじめに

整数の乗算命令、つまり N ビットの整数 m, n から $2N$ ビットの積 $m*n$ を求める命令は、近年のマイクロプロセッサではかなり高速に実行できるようになった。それに比べて、整数の除算命令、つまり、N ビットの整数 n と d から、整数に切り捨てた商 $q = \lfloor n/d \rfloor$ を求める命令は、数倍の時間がかかるのが普通である。

除数が定数とわかっている場合には、その逆数に相当する数をあらかじめ計算しておけば、乗算命令を含む命令列を使って、除算を実行することができ、除算を高速化することができる。ただし、除算命令を使った結果と完全に一致させるためには、いくつかの工夫が必要になる。

本発表では、乗算を使って除算を正確に実行するための二つの基本式について、それが適用できる必要十分条件、およびその証明を示す。また、それを利用して、二つの基本式から派生する計算方法の中で、与えられた除数について最良のものを選ぶ方法を示す。

以下本発表報告では、2節で既存の方法を紹介し、その問題点を指摘する。次に3節で、本発表の方法を説明する。それがきちんと機能することの証明は4節で行う。5節では、既存の方法を含めたコード例を用いて、評価を行う。最後に6節でまとめを行う。

2　従来の方法の問題点

[Granlund94] で提案され、GCC に実装されている方法は、どの除数に関しても一つの基本式を使って計算しているため、そのままプロセッサの命令に直すと、定数の値や中間結果が、プロセッサの扱えるビット数を超えてしまう場合がある。そのため、複雑な命令列を使ってプロセッサの扱えるビット数に収める必要がある。

[Granlund94] ではまた、除数がある条件を満たす場合に、複雑な命令列を単純化する方法も述べられている。しかしそれは、ad hoc なものであり、系統的な必要十分条件とはなっていない。

[Fog96] の方法は、本発表と同様の二つの基本式を使っているが、式の使える条件について述べられておらず、系統的なテストが必要と指摘するにとどまる。従って、利用するためには、許されるすべての被除数について、正しい商を生成することを確認する必要がある。

[Warren02] は、[Granlund94] と同様に、どの除数に関しても、一つの基本式を使って計算している。ただし、その適用条件については、必要十分条件を与えている。その条件は、本発表の結果と本質的に同じである (付録 A で同値であることを示す)。[Warren02] には、必要十分条件となっていることの証明も述べられているが、たいそう複雑である。

[AMD03] には、除数を与えると、除算を行うアセンブリ言語プログラムを出力するような、C 言語のプログラムが紹介されている。これは二つの基本式を使っている。C 言語プログラム中で使っている判定は、必要十分条件になっているが、その証明は [AMD03] にはない (本発表の結果と同値であることを付録 A で示す)。

3　本発表の方法

本節では、本発表の方法による、除算を使った定数除算の実現方法について述べる。

プロセッサの命令の働きを式で表すため、次の表記を用いる。$\mathrm{MULUH}(m,n)$ は、m と n の符号なしの積（$2N$ ビットの数）の上位 N ビットを表し、$\mathrm{SRL}(n,b)$ は、n を右に b ビット論理シフトしたものを表す。

N ビット符号なし数に関して、整数除算 $q = \lfloor n/d \rfloor$ を行う命令列を求めたいとする。ただし、$0 < d < 2^N$ は整定数で、$0 \le n < t$ は整数の変数である（通常は $t = 2^N$ とする）。d は 2 の冪ではないとする。$k = \lfloor t/d \rfloor$、$k' = \lfloor (t-1)/d \rfloor$ とおく（$t = 2^N$ なら $k = k'$ である）。

0 以上の整数 b を適当に選ぶ（適当かどうかの判定方法をこれから述べる）。$M = 2^{N+b}$ とし、$m = \lceil M/d \rceil$、$m' = \lfloor M/d \rfloor$ とする。除算を乗算によって計算するための二つの基本式

$$\mathrm{SRL}(\mathrm{MULUH}(m,n),b) \tag{A}$$

$$\mathrm{SRL}(\mathrm{MULUH}(m',n+1),b) \tag{B}$$

が使える必要十分条件は、

$$\forall n\ \mathrm{SRL}(\mathrm{MULUH}(m,n),b) = \lfloor n/d \rfloor \ \Leftrightarrow\ m > k(dm - M)$$

$$\forall n\ \mathrm{SRL}(\mathrm{MULUH}(m',n+1),b) = \lfloor n/d \rfloor \ \Leftrightarrow\ m' \ge k'(M - dm')$$

である。そこで、目的のプロセッサのアーキテクチャにより、b のなるべく小さくなる式、あるいは、加算の回数の少ない基本式 (A) を選べばよい。

基本式 (B) は、$n+1$ を用いているため、$n = 2^N - 1$ の場合にうまく計算できず、場合分けが必要になる。実際の命令列に直す場合には、このことを考慮する必要がある。

基本式 (A) から派生する計算方法としては、d が偶数の場合に、d を $d = 2^e d'$ となる d' で置き換えて m を求め、q を

$$\mathrm{SRL}(\mathrm{MULUH}(m, \mathrm{SRL}(n, e)), b) \tag{C}$$

で計算するものがある。基本式 (A) が適用できない場合でも、派生式 (C) が適用できる場合がある。基本式 (B) で場合分けを含む加算を使うより、論理シフトのほうが望ましい場合、派生式 (C) を使う価値がある。

ここで、計算式の選び方の一例を示す。基本式 (B) より派生式 (C) が望ましく、派生式 (C) より基本式 (A) が望ましく、同じ式の中では b が小さい方が望ましい場合のものである。d が与えられたとき、

- $b = 0, 1, 2, \cdots, \lfloor \log_2 d \rfloor$ に対して (A) のための必要十分条件が成り立つか調べ、最初に見つかった b について (A) を利用する。見つからなければ次に進む。
- d が奇数なら次に進む。偶数なら $b = 0, 1, 2, \cdots, \lfloor \log_2 d \rfloor$ に対して (C) のための必要十分条件が成り立つか調べ、最初に見つかった b について (C) を利用する。見つからなければ次に進む。
- $b = 0, 1, 2, \cdots, \lfloor \log_2 d \rfloor$ に対して (B) のための必要十分条件が成り立つか調べ、最初に見つかった b について (B) を利用する。

$b = \lfloor \log_2 d \rfloor$ については、基本式 (A) または (B) のどちらかが必ず適用可能なので、この方法で必ずどれかの計算式が決まる。

なお、本発表では、目的のプロセッサが乗算命令、加算命令、論理シフト命令を持っていることを前提としているが、乗算命令に関しては、加算命令とシフト命令の組合せによって実現することもできる。

4 本発表の方法の証明

本節では、前節で紹介した方法が正しく機能することを証明する。最初に簡単な補題を証明し、それを用いて、基本式 (A) の使える必要十分条件を示す。次に、基本式 (B) の使える必要十分条件を示す。最後に、どんな除数であっても、基本式 (A) と (B) の少なくとも一つが使えることを示す。

4.1 補題

証明すること

一次関数 F が、$F(-1) \geq -M$ かつ $F(0) < 0$ を満たせば、$F(1) < M$ である。

証明

$$F(x) = F(-1) + \frac{F(0) - F(-1)}{0 - (-1)}(x - (-1))$$

と書けるので、

$$F(1) = F(-1) + \frac{F(0) - F(-1)}{0 - (-1)}(1 - (-1)) = 2F(0) - F(-1) < M$$

となる。

4.2 基本式 (A) の必要十分条件の証明

前提

次のように整数 N, d, t, b が与えられているとする。

- $N > 0$
- $0 < d < 2^N$ かつ、d は 2 の冪ではない
- $0 < t \leq 2^N$
- $b \geq 0$

証明すること

$M = 2^{N+b}$、$k = \lfloor t/d \rfloor$、$k' = \lfloor (t-1)/d \rfloor$、$m = \lceil M/d \rceil$ とおく。すると、

$$\text{すべての } n = 0, 1, \cdots, t-1 \text{ について、} \mathrm{SRL}(\mathrm{MULUH}(m, n), b) = \lfloor n/d \rfloor \tag{A1}$$

が成り立つ必要十分条件は、$m > k(dm - M)$ である。

証明

(A1) は (A2),(A3),(A4) がすべて成り立つことと同値である。

$$\text{すべての } j = 0, 1, \cdots, k' \text{について、} jM \leq jdm \tag{A2}$$

$$\text{すべての } j = 1, 2, \cdots, k \text{ について、} (jd-1)m < jM \tag{A3}$$

$$kd < t \text{ のとき、} (t-1)m < (k+1)M \tag{A4}$$

そのため、(A2),(A3),(A4) がすべて成り立つ必要十分条件が $m > k(dm - M)$ であることを示せばよい。

$m = \lceil M/d \rceil \geq M/d$ なので、$dm \geq M$ であることを、以後の証明で用いる。

$M \leq dm$ なので、$jM \leq jdm$ は任意の負でない整数 j について成り立つ。よって、(A2) は常に成り立つ。

(A3) の両辺の差を求めると、$jM - (jd-1)m = m - j(dm - M)$ となる。$dm - M \geq 0$ なので、(A3) の成り立つ必要十分条件は、$m - k(dm - M) > 0$ である。

$kd < t$ のとき、$t - 1 = kd + r$ と表すと、$r \leq d - 2$ である。そこで、$(kd + d - 2)m < (k+1)M$ を示せば、(A4) を示したことになる。

$F(x) = (kd - 1 + (d-1)x)m - kM$ とする。F は一次関数である。(A2),(A3) が成り立つと仮定すれば、

$$F(-1) = (k-1)dm - kM \geq (k-1)M - kM = -M \text{ より } F(-1) \geq -M$$

$$F(0) = (kd-1)m - kM < kM - kM = 0 \text{ より } F(0) < 0$$

である。補題により、$F(1) < M$ となる。$F(1) = (kd + d - 2)m - kM$ なので、$(kd + d - 2)m < (k+1)M$ となる。すなわち、(A2),(A3) が成り立つと仮定すれば、(A4) が成り立つ。

以上をまとめて、必要十分条件であることを示す。(A2),(A3),(A4) が成り立っているとすれば、(A3) より $m > k(dm - M)$ が成り立つ。逆に $m > k(dm - M)$ が成り立っているとすれば (A3) が成り立ち、常に成り立つ (A2) と合わせて (A4) が成り立つので、(A2),(A3),(A4) がすべて成り立つ。これで証明が完了した。

4.3　基本式 (B) の必要十分条件の証明

前提

次のように整数 N, d, t, b が与えられているとする。

- $N > 0$
- $0 < d < 2^N$ かつ、d は 2 の冪ではない
- $0 < t \leq 2^N$
- $b \geq 0$

証明すること

$M = 2^{N+b}$、$k = \lfloor t/d \rfloor$、$k' = \lfloor (t-1)/d \rfloor$、$m' = \lceil M/d \rceil$ とおく。すると、

$$\text{すべての } n = 0, 1, \cdots, t-1 \text{ について、} \mathrm{SRL}(\mathrm{MULUH}(m', n+1), b) = \lfloor n/d \rfloor \tag{B1}$$

が成り立つ必要十分条件は、$m' \geq k'(M - dm')$ である。

証明

(B1) は (B2),(B3),(B4) がすべて成り立つことと同値である。

$$\text{すべての } j = 0, 1, \cdots, k' \text{について、} jM \leq (jd+1)m' \tag{B2}$$

$$\text{すべての } j = 1, 2, \cdots, k \text{ について、} (jd-1+1)m' < jM \tag{B3}$$

$$kd < t \text{ のとき、} (t-1+1)m' < (k+1)M \tag{B4}$$

そのため、(B2),(B3),(B4) がすべて成り立つ必要十分条件が $m' \geq k'(M - dm')$ であることを示せばよい。

$m' = \lfloor M/d \rfloor \leq M/d$ であり、d は 2 の冪でないので、$dm' < M$ であることを、以後の証明で用いる。

(B2) の両辺の差を求めると、$(jd+1)m' - jM = m' - j(M - dm')$ となる。$M - dm' > 0$ なので、(B2) の成り立つ必要十分条件は、$m' - k'(M - dm') \geq 0$ である。

$dm' < M$ なので、$jdm' < jM$ は任意の正の整数 j について成り立つ。よって、(B3) は常に成り立つ。

$tm' \leq (k+1)dm' < (k+1)M$ なので、(B4) は常に成り立つ。

以上により、(B1) が成り立つ必要十分条件が $m' \geq k'(M - dm')$ であることが示され、証明が完了した。

4.4　基本式 (A) と (B) の少なくとも一つが使えることの証明

前提

次のように整数 N, d が与えられているとする。

- $N > 0$
- $0 < d < 2^N$ かつ、d は 2 の冪ではない

b を、$2^b \leq d < 2^{b+1}$ となる整数とする。

本発表	[Granlund94]	[Fog96]	[Warren02]	[AMD03]
A	A	A	A	A
A	C/D	B	A	A
C	C	B	D	B’
B	D	B	D	B’

表 1: 各方法で使われる計算式

証明すること

$k = \lfloor 2^N/d \rfloor = \lfloor (2^N - 1)/d \rfloor$、$m = \lceil 2^{N+b}/d \rceil$、$m' = \lceil 2^{N+b}/d \rceil$ とおく。すると、以下の (1) と (2) の少なくとも一つが成り立つ。

$$0 < m < 2^N \quad \text{かつ} \quad m > k(dm - 2^{N+b}) \tag{1}$$

$$0 < m' < 2^N \quad \text{かつ} \quad m' \geq k(2^{N+b} - dm') \tag{2}$$

証明

$M = 2^{N+b}$、$f = M/d$ とする。$M = 2^N 2^b \leq 2^N d$ なので、$f \leq 2^N$ である。$m, m' \leq 2^N$ となるので、(1)(2) の前半の条件が成り立つ。ここで、f の小数部によって場合分けを行う。

- f が整数 ($f = \lfloor f \rfloor$) の場合は、d は M の約数、つまり 2 の冪であり、前提に反するので、これはありえない。
- $f - \lfloor f \rfloor = 0.5$ の場合は、d は $2M$ の約数、つまり 2 の冪であり、同様にありえない。
- $0 < f - \lfloor f \rfloor < 0.5$ の場合は、(2) が成り立つことを示す。

  $f = M/d > M/2^{b+1} = 2^{N-1}$ なので、$m' \geq 2^{N-1}$ である。また、$k(M - dm') = kd(f - m') < 2^N * 0.5 = 2^{N-1}$ である。つまり、(2) の後半が成り立つ。
- $0.5 < f - \lfloor f \rfloor < 1$ の場合は、(1) が成り立つことを示す。

  $f = M/d > M/2^{b+1} = 2^{N-1}$ なので、$m > 2^{N-1}$ である。また、$k(dm - 2^{N+b}) = kd(m - f) < 2^N * 0.5 < 2^{N-1}$ である。つまり、(1) の後半が成り立つ。

以上により、(1) と (2) の少なくとも一つが成り立つことが示され、証明が完了した。

5 評価

本節では、本発表の方法と、他の既存の方法を比較して評価する。評価には、Intel x86 プロセッサの 32 ビットモードでのコードを用いている。

本発表の基本式 (A) と (B)、派生式 (C) に対応するコードの例を、図 1 の (A)、(B)、(C) に示す。基本式 (B) には、被除数 n が最大値のとき、$n + 1$ がオーバーフローするという問題があるため、条件ジャンプを使って場合分けを行っている。$n + 1 = 2^{32}$ なので、EDX に入れた乗数は、$n + 1$ との積の上位に等しいことに注意されたい。被除数が最大値になることはまれなので、分岐予測を行うプロセッサでは条件ジャンプのペナルティーは小さいと予想される。

表 1 は、本発表の方法と、他の方法で、(A)(B)(B’)(C)(D) の 5 つの式をどのような場合分けで使い分けているのかを、比較したものである。各式については以下で説明する。

(A) d=5 q=SRL(MULUH(3435973837,n),2)

```
MOV EDX,0CCCCCCCDH
MUL EDX
SHR EDX,2
```

(B) d=7 q=SRL(MULUH(1227133513,n+1),1)

```
    MOV EDX,49249249H
    INC EAX
    JZ  L1
    MUL EDX
L1: SHR EDX,1
```

(B') d=7 q=SRL(HIGH(MULU(1227133513,n)+1227133513),1)

```
MOV EDX,49249249H
MUL EDX
ADD EAX,49249249H
ADC EDX,0
SHR EDX,1
```

(C) d=14 q=SRL(MULUH(2454267027,SRL(n,1)),2)

```
SHR EAX,1
MOV EDX,92492493H
MUL EDX
SHR EDX,2
```

(D) d=7 t1=MULUH(613566757,n) q=SRL(t1+SRL(n-t1,1),2)

```
MOV EDX,24924925H
MOV EAX,ECX
MUL EDX
SUB EDX,ECX
SHR ECX,1
LEA EAX,[ECX+EDX]
SHR EAX,2
```

図 1: Intel x86 でのコード例

[Granlund94] の方法は、本発表の基本式 (B) の代わりに、基本式 (A) を変形した式を使っている。その式とコードの例を図 1(D) に示す。また、判定条件が必要十分条件になっていないので、本発表の方法で (A) を使える場合でも、(C) または (D) を使ってしまうことがある。

[Fog96] の方法は、基本式 (A) と (B) だけを使う方法である。やはり、判定条件が必要十分条件になっていないので、本発表の方法で (A) を使える場合でも (B) を使ってしまうことがある。なお、[Fog96] では、被除数の範囲が十分狭いときには、乗算の上位 N ビットではなく下位 N ビットが使えることが指摘されている。

[Warren02] の方法は、基本式 (A) と、[Granlund94] と同じ式 (D) を使っている。判定条件は必要十分条件である。

[AMD03] の方法は、基本式 (B) の代わりに、64 ビット加算を使った式を使っている。その式とコードの例を図 1(B') に示す。ここで、MULU(m,n) は、m と n の符号なしの積（$2N$ ビットの数）を、HIGH(p) は、$2N$ ビットの数 p の上位 N ビットを表す。この式は、条件分岐が不要になる代わりにキャリーつき加算を必要とする。[AMD03] の判定条件は、必要十分条件になっている。

式	x86でのバイト数	Pentium III	Pentium 4
A	10	6	20
B	12	6	16
B'	17	8	20
C	12	8	16
D	19	9	24

表 2: 各コード例のクロック数

表2は、Pentium IIIおよびPentium 4プロセッサで、RDTSC命令を使って測定した、各コード例のクロック数である。分岐予測が働くように、同じ被除数について10回実行したうちの10回目を測定している。Pentium IIIプロセッサでは、予想されるようなクロック数の差が認められる。Pentium 4プロセッサの場合は、差が不明瞭であり、更なる調査が必要である。

以上をまとめると、本発表の方法は、既存の方法のいいところをとって、最良の命令列を生成する方法を与えているといえる。

6 まとめ

本発表では、Nビットの符号なし除算$\lfloor n/d \rfloor$を行う二つの基本式について、それが使用可能な必要十分条件を示した。除数dと、被除数の最大値+1であるtが与えられたとき、適当なbを選ぶための条件は、$M = 2^{N+b}$、$m = \lceil M/d \rceil$、$m' = \lfloor M/d \rfloor$、$k = \lfloor t/d \rfloor$、$k' = \lfloor (t-1)/d \rfloor$とするとき、

$$\forall n \ \mathrm{SRL}(\mathrm{MULUH}(m, n), b) = \lfloor n/d \rfloor \ \Leftrightarrow \ m > k(dm - 2^{N+b})$$

$$\forall n \ \mathrm{SRL}(\mathrm{MULUH}(m', n+1), b) = \lfloor n/d \rfloor \ \Leftrightarrow \ m' \geq k'(2^{N+b} - dm')$$

である。

本発表ではまた、それが必要十分条件であることの証明と、二つの基本式の少なくとも一つが適用可能であることの証明を与えた。

2番目の基本式を使用する場合は、被除数が最大のときのために場合分けが必要になるが、条件分岐を使ったとしても分岐確率が偏っているため、分岐予測を行うプロセッサでは良好な性能を期待できる。本発表では実際の命令列のクロック数を測定して、特にIntelのPentium IIIプロセッサの場合に、本発表の方法が最良の命令列を生成していることを示した。

参考文献

[Granlund94] Torbjörn Granlund and Peter L. Montgomery. Division by Invariant Integers using Multiplication. In *Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation*, pp. 61–72, 1994. (`http://www.swox.com/~tege/`)

[Fog96] Agner Fog. *How to optimize for the Pentium family of microprocessors.* 1996-2004. (`http://www.agner.org/assem/`)

[Warren02] Henry S. Warren, Jr. *Hacker's Delight.* Addison-Wesley, 2002. (日本語訳: ハッカーのたのしみ. エスアイビー・アクセス, 2004)

[AMD03] *Software Optimization Guide for AMD Athlon™ 64 and AMD Opteron™ Processors*. Advanced Micro Devices, Inc., 2003. (`http://www.amd.com/jp-ja/Processors/TechnicalResources/0,,30_182_739_7203,00.html`)

A 他の既存の方法と本発表の方法で、判定条件が同値であることの証明

[Warren02] の方法と同値であることは、次のようにして示すことができる。まず、[Warren02] の 10 章の表記と本発表の表記は、次のように対応している。

$$\begin{aligned} n_c &\rightarrow kd-1 \\ 2^p &\rightarrow M \\ d-1-\mathrm{rem}(2^p-1,d) &\rightarrow dm-M \end{aligned}$$

すると、[Warren02] の 181 ページの式 (27) は、次のように変形でき、本発表の方法と同値であることがわかる。

$$\begin{aligned} 2^p > n_c(d-1-\mathrm{rem}(2^p-1,d)) &\Leftrightarrow M > (kd-1)(dm-M) \\ &\Leftrightarrow M > kd(dm-M)+M-dm \\ &\Leftrightarrow dm > kd(dm-M) \\ &\Leftrightarrow m > k(dm-M) \end{aligned}$$

[AMD03] の方法と同値であることも、同じように示せる。188 ページから始まるプログラム例では、190 ページで次のように変数を設定している。

$$\begin{aligned} \texttt{m\_low} &= \lfloor 2^{32+\texttt{l}}/\texttt{t} \rfloor \\ \texttt{j} &= \mathrm{rem}(2^{32}-1,\texttt{t}) \\ \texttt{k} &= \lfloor 2^{32+\texttt{l}}/(2^{32}-1-\texttt{j}) \rfloor \\ \texttt{m\_high} &= \lfloor (2^{32+\texttt{l}}+\texttt{k})/\texttt{t} \rfloor \end{aligned}$$

これらと本発表の表記は、次のように対応する。

$$\begin{aligned} \texttt{m\_low} &\rightarrow m-1 \\ \texttt{t} &\rightarrow d \\ 2^{32}-1-\texttt{j} &\rightarrow kd \\ 2^{32+\texttt{l}} &\rightarrow M \end{aligned}$$

190 ページの while 文では、`m_low<m_high` を満たす限界まで `l` を最小化しているので、これが判定条件となる。変形していくと、本発表の方法と同値であることがわかる。

$$\begin{aligned} \texttt{m\_low} < \texttt{m\_high} &\Leftrightarrow m \le \lfloor (M+\lfloor M/kd \rfloor)/d \rfloor \\ &\Leftrightarrow dm \le M+\lfloor M/kd \rfloor \\ &\Leftrightarrow^* dm < M+M/kd \\ &\Leftrightarrow k(dm-M) < M/d \\ &\Leftrightarrow^* k(dm-M) < m \quad \because m = \lceil M/d \rceil \end{aligned}$$

ここで、$\Leftrightarrow^*$ は d が 2 の冪でないことを利用した変形である。

B　質疑応答 (敬称略)

八杉 基本式 (A) か (B) かは何に依存するのか

著者 除数だけに依存する (コンパイル時に決定)

粕川 基本式 (B) になる除数の除算をたくさん使った場合に、分岐履歴がはみ出してしまうことがあるのではないか。

著者 AMD のプロセッサでは 8192 エントリあるので、問題になることはあまりないと思う。Intel プロセッサでも増えてきている。[1]

和田 気持ちは判るが、良く出てくる除数は表で良いじゃない。2,3,4 は頻出だけど 31 は少ない。

著者 除数が実行時定数の場合は、表を使って高速化する意味はある。ここではコンパイル時定数を考えている。

和田 余りが出ないのはどう思う?

著者 考えてみたが、[Warren02] にもあるように、商に除数をかけて被除数から引く以外に、うまい方法が見つからない

上田 乗算命令と除算命令の速さはそんなに違うのか

著者 乗算命令は 3-10 クロック程度だが、除算命令は 20 クロック程度が普通

粕川 実行ユニットのコンフリクトで、除算命令を使ったほうが速いことはないのか

著者 除算命令はパイプライン化されてなくて、ほかの命令もストールしてしまうプロセッサが多い

和田 やっぱり余りが欲しい

紀 (座長) 暗号だと多倍長が良く使われるが、そういう話はないか

著者 除数が多倍長でなくていいなら同じ方法が使えるが、一般性はない

和田 その場合は余りが必要

和田 3 で割ってよ。商が大きくなりうるので一番苦しい。1/3 は 010101 $\cdots$

大駒 シフトを使うんですか?

著者 ビットパターンの繰り返しを使ってうまくやる

和田 等差級数の和を求めるには除算を使うでしょ

著者 定数掛け算は乗数の log オーダーだけど、ビット数の log くらいになるかもしれない

粕川 Pentium 4 で結果が妙だったのは、命令を変換したりトレースキャッシュを使ったりしているので、しかたないのではないか。演算の組み合わせで速くなったり遅くなったりする。

著者 実行タイミングを調査した文書はあるが、まだきちんとした測定環境を作れていない。

八杉 コメントだけど 東大の加藤さんの循環小数でなくて整数部が無限にある数を使うと割り算が掛け算になる?

和田 それ、p-adic number だよね。関係ない。

八杉 数学と違って有限だから何か近道があるかもしれない。

[1] AMD の K6 は 8192 エントリ、Athlon は 2048 エントリ、Intel の Pentium III は 512 エントリ、Pentium 4 は 4096 エントリとなっている。