高速な数式処理ソフトウェアを作ってみよう

木村 欣司 (京都大学大学院情報学研究科)

講演内容

・高速に計算を行う数式処理ソフトウェアを作るにはどうしたらよいかを話す

数式処理ソフトウェアの話題

キャッシュのお話

キャッシュとは

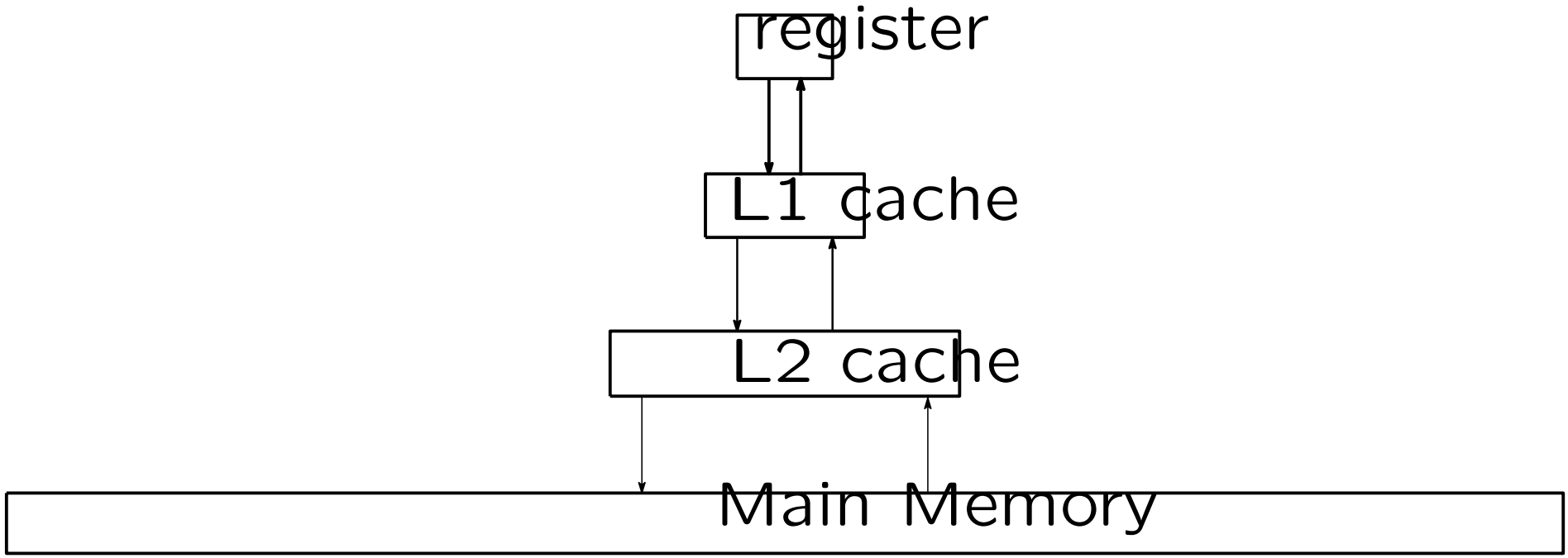
1.CPUの内部にあり，メインメモリよりもはるかに容量が小さい

2.最近読み込んだデータとその周辺のデータ，あるいは，最近計算した結果，あるいは最近使った命令列とその周辺の命令列を，一時的に蓄えておくところ，使われなくなったデータは，メインメモリに返される

3.メインメモリと registerの間よりも，キャッシュとregisterの間のほうが，単位時間当りのデータを移動できる量が多い

4.L1，L2，(L3)，(L4)の階層構造になっている

キャッシュ階層



例外として，メインメモリから，L2キャッシュを飛び越えて，L1キャッシュにデータが直接入るCPUがある

L2キャッシュから，L1キャッシュを飛び越えて，レジスタにデータが直接入るCPUがある

キャッシュ階層からみた優れたアルゴリズム

ベクトルの内積

・データの再利用性なし

行列ベクトル乗算, Rank-1 update

・ベクトル側にデータの利用性あり, 行列側にはなし

行列行列乗算

・データの利用性大いにあり

⇒ **可能な限り行列行列乗算を中心としたアルゴリズムを設計するべし**

two-stage algorithmの話

行列行列乗算による高速化

実対称行列の固有値問題

Aを$n \times n$の与えられた実対称行列とする,

$$AV = VD, \quad V^\top V = VV^\top = I_n,$$
$$D = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}, \; V = \begin{pmatrix} v_1 & \cdots & v_n \\ & & \end{pmatrix}$$

となる, 固有値$\lambda_i (1 \leq i \leq n)$, 固有ベクトル$V(n \times n)$を計算する

実対称行列Aの固有値分解の解法手順

1. 固有値保存変形により，$A \rightarrow$ 実対称3重対角行列T

(1)Householder変換(Dongarra法，行列とベクトルの乗算を含む)

(2)Bischof/Wuのアルゴリズム＋村田法

2. 以下の問題を解く

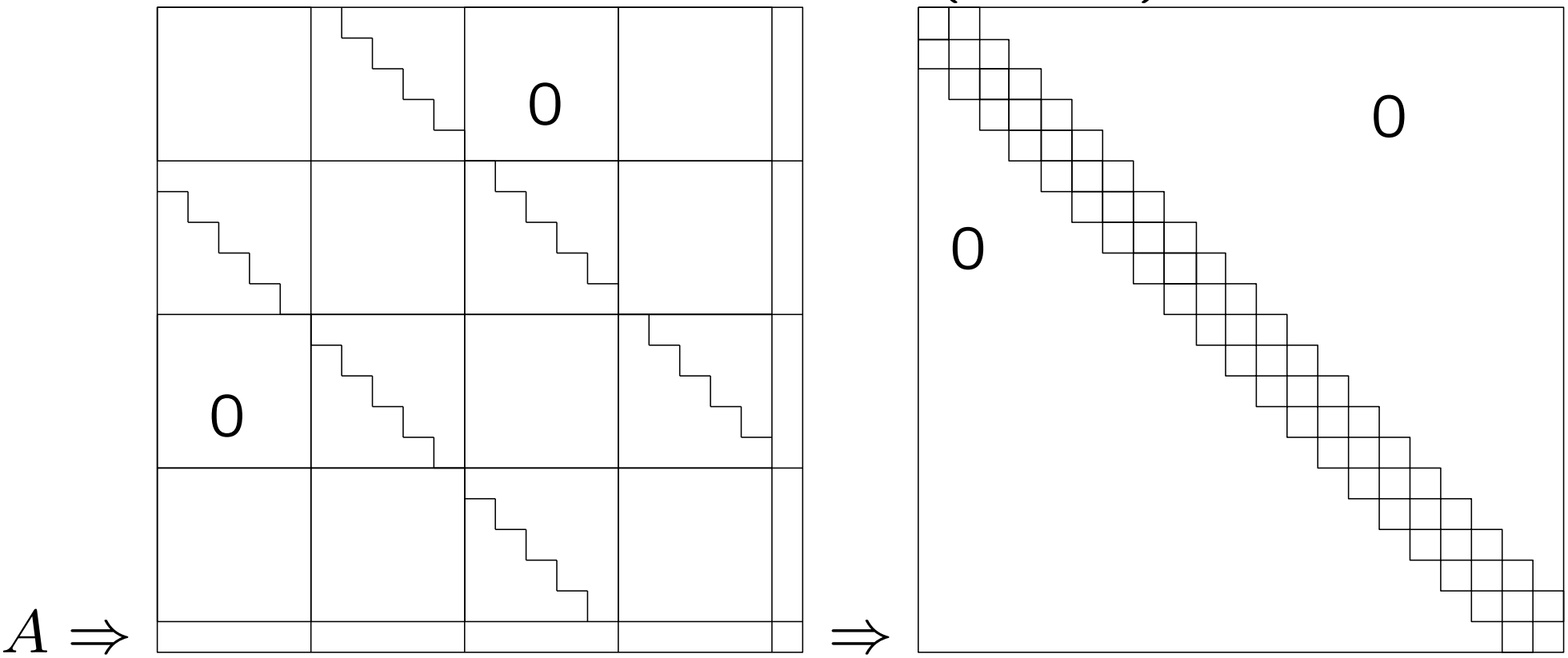
$$TV' = V'D, \quad V'^{\top}V' = V'V'^{\top} = I_n$$

3. 逆変換によりV'からVを得る

Bischof/Wuのアルゴリズム＋村田法

$n \times n$の実対称行列Aを，**ほぼ行列行列乗算のみ**を用いて帯行列に変換し(Bischof/Wuアルゴリズム)，

帯行列を３重対角行列Tに変換する(村田法)

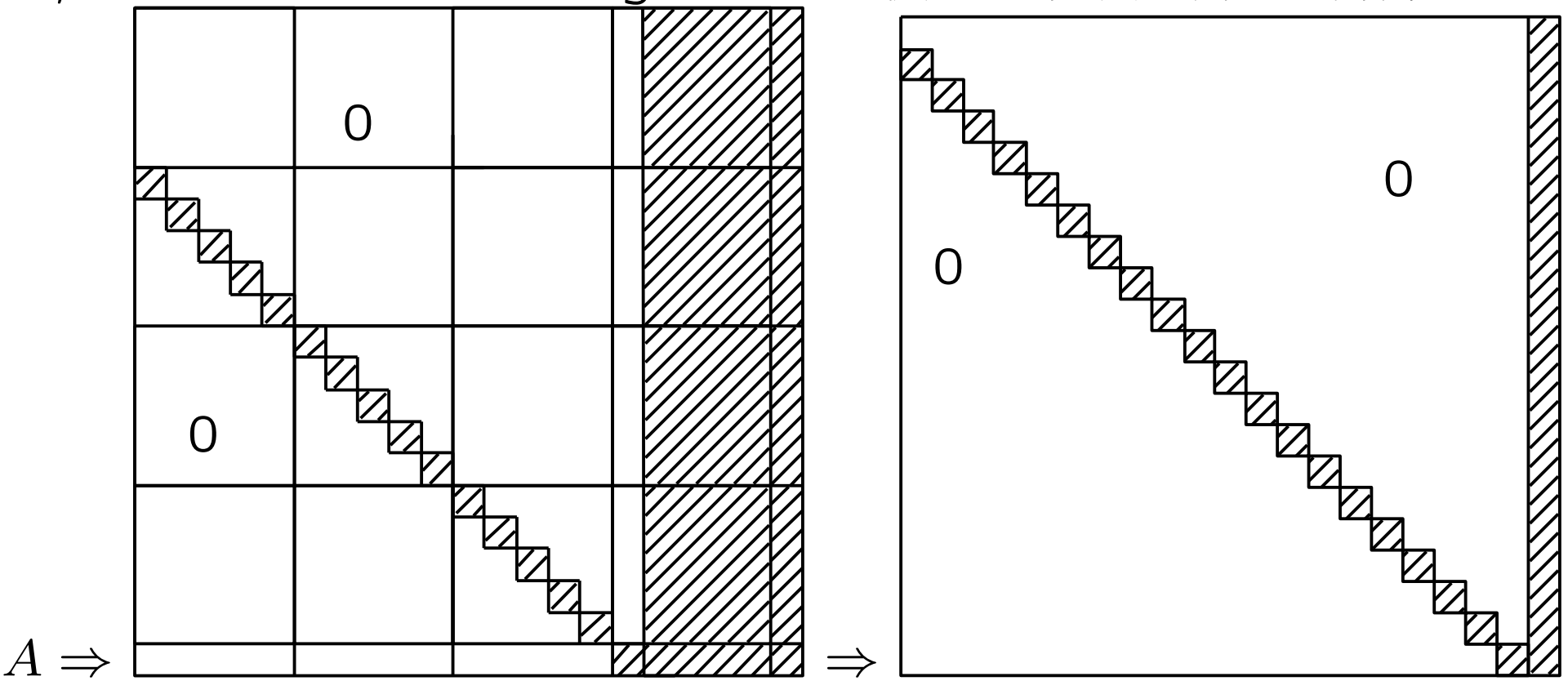


数式処理における固有多項式を計算するための two-stage algorithm

有限体$\mathbb{Z}/p\mathbb{Z}$上において，$N \times N$の非対称**密行列**Aの固有多項式を計算する，

固有多項式の次数＝最小多項式の次数を仮定，

$N \times N$の非対称行列Aを，行列乗算を利用して帯コンパニオン行列Cに変換し，CからWiedemann algorithmを使って最小多項式を計算する



タイミングデータ

$N \times N$の整数行列の固有多項式の計算時間の比較

各要素は，[1,10]の整数

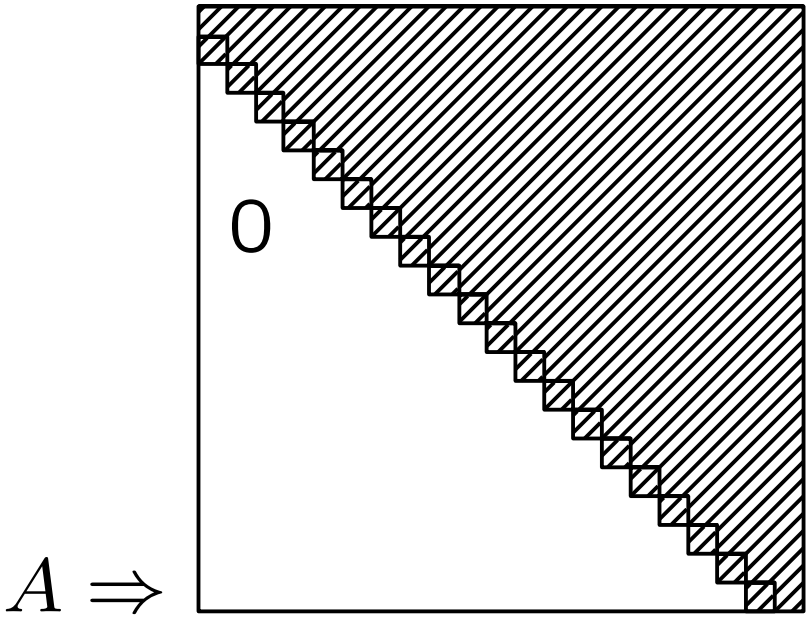
N	one-stage	two-stage
1000	13.81sec	9.73sec
2000	3m41.18sec	48.37sec
3000	17m48.96sec	3m20.75sec

実験環境:後述のCRAY XC30 1node，GCC 4.9.2，Intel MKL

固有多項式を計算するためのone-stage algorithm

有限体$\mathbb{Z}/p\mathbb{Z}$上において，$N \times N$の非対称密行列Aの固有多項式を計算する，**固有多項式の次数＝最小多項式の次数の仮定は必要ない**

クリロフ部分空間法を利用する，ただし，直接法とも解釈できる方法を使う，計算の大部分を，行列ベクトル乗算・Rank-1 updateが占める



中国剰余定理による数式処理ソフトの高速化の話

中国剰余定理

未知の整数をXとする

$$X \bmod 2 = 1$$

$$X \bmod 3 = 2$$

この情報から，**中国剰余定理**により

$$X = \cdots, -13, -7, -1, 5, 11, 17, \cdots$$

が，Xの候補となる，解は一意に定まらない

もし，Xは1桁の整数であるという付加情報が手に入ったとすると，$X = -7, -1, 5$，まだ，3つも候補が残っている

もうすこし情報が手に入ったとする

$$X \bmod \underline{2} = 1$$
$$X \bmod \underline{3} = 2$$
$$X \bmod \underline{5} = 3$$

よって，答えは，中国剰余定理と付加情報により$X = -7$と一意的に定まる，**下線の部分には，素数を使う**

(付加情報) 行列式に付随した2つの上界公式

2つのノルムを定義する: qは, 多変数多項式

$$\|q\|_1 = \sum_{\alpha_1,\cdots,\alpha_s} |c(\alpha_1,\cdots,\alpha_s)|,$$
$$\|q\|_2 = \sqrt{\sum_{\alpha_1,\cdots,\alpha_s} c(\alpha_1,\cdots,\alpha_s)^2},$$

ここで,

$q = \Sigma_{\alpha_1,\cdots,\alpha_s} c(\alpha_1,\cdots,\alpha_s) x_1^{\alpha_1}\cdots x_s^{\alpha_s} \in \mathbb{Z}[x_1,\cdots,x_s]$

$A = (a_{i,j}) \in \mathbb{Z}^{N\times N}$ についてのHadamardの上界は,

$$|\det(A)| \leq \min\left\{ \prod_{i=1}^{N} \sqrt{\sum_{j=1}^{N} |a_{i,j}|^2}, \prod_{j=1}^{N} \sqrt{\sum_{i=1}^{N} |a_{i,j}|^2} \right\}$$

$A = (a_{i,j}) \in \mathbb{Z}[x_1, \cdots, x_s]^{N\times N}$ についてのGoldsteinとGrahamの上界は,

$$\|\det(A)\|_2 \leq \min\left\{ \prod_{i=1}^{N} \sqrt{\sum_{j=1}^{N} \|a_{i,j}\|_1^2}, \prod_{j=1}^{N} \sqrt{\sum_{i=1}^{N} \|a_{i,j}\|_1^2} \right\}$$

固有多項式の係数の上界もこの式から得られる

$$B = \begin{vmatrix} a & 2b \\ 3c & 4d + f \end{vmatrix} = \underline{4}ad + \underline{1}af - \underline{6}bc$$

G.&G.の上界$= \min(\sqrt{1+4}\sqrt{9+25}, \sqrt{1+9}\sqrt{4+25}) <$ 13.1

固有多項式の中国剰余定理による計算法

整数行列Aの固有多項式$f(x) = \det(xI - A)$を計算

$$\begin{aligned} f_1(x) &= f(x) \bmod p_1 \\ f_2(x) &= f(x) \bmod p_2 \\ &\vdots \end{aligned}$$

中国剰余定理により，$f(x)$を復元

有限体$\mathbb{Z}/p\mathbb{Z}$の実装の話

技法.1:C言語のGNU拡張を使う方法

$p, a, b \in \mathbb{Z}, p < 2^{63}, 0 \leq a, b < p, a \oplus b = (a + b) \bmod p$と定義する

$0 \leq a_0, \cdots, a_{99999999} < p$について, 総和を考える

$$r = a_0 \oplus a_1 \oplus \cdots \oplus a_{99999999}$$

間違った実装の方法

```
unsigned long long a[1000000],r;
r=0;
for(i=0;i<1000000;i++)
    r=(r+a[i]) % p;
```

正しい実装の方法

$$
\begin{aligned}
r &= a_0 \oplus a_1 \oplus \cdots \oplus a_{99999999} \\
&= (a_0 + a_1 + \cdots + a_{99999999}) \bmod p
\end{aligned}
$$

```
typedef unsigned int uint128_t __attribute__((mode(TI)));
unsigned long long a[1000000],r;
uint128_t t=0;
for(i=0;i<1000000;i++) t+=a[i];
r=t % p;
```

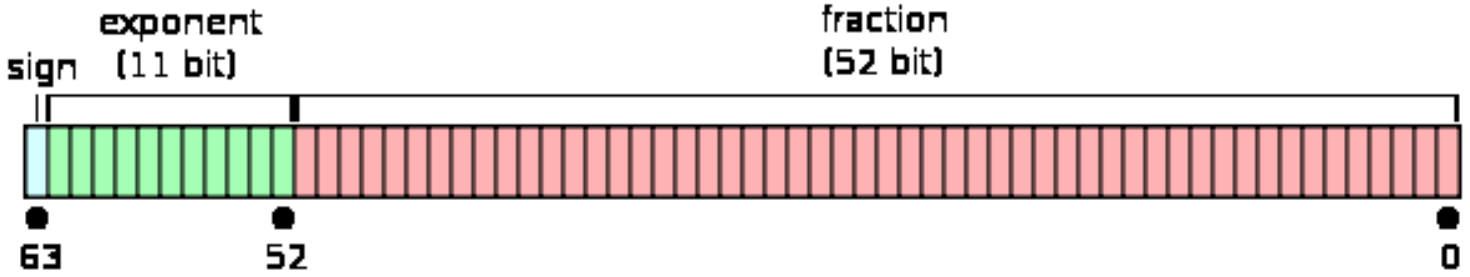
技法.2:浮動小数点数を使う方法

浮動小数点数:

$$\text{正規化数の場合} \quad \pm 1.x_1x_2\cdots \times 2^{y_1y_2\cdots - M}$$

・単精度浮動小数点数

・倍精度浮動小数点数



Basic Linear Algebra Subprograms（BLAS）：単精度・倍精度浮動小数点数のベクトルと行列に関する基本線型代数操作ライブラリ

倍精度浮動小数点数：連続して表現可能な整数の範囲，-1.0×2^{53} から 1.0×2^{53}

代表元を，$-\frac{p-1}{2}$ から $\frac{p-1}{2}$ の中で選ぶことにすると，行列サイズを N とするとき，内積の絶対値最大に対する条件は，

$$N\left(\frac{p-1}{2}\right)^2 \leq 2^{53}$$

$$p \leq 2\sqrt{\frac{2^{53}}{N}} + 1$$

この不等式を満たす p を採用すれば，BLAS を利用可能

Xeon Phiを活用するための準備

行列ベクトル乗算・Rank-1 updateなどでは，CPUとメモリの間のデータの移動がボトルネック

メモリ空間には，単精度浮動小数点数を用いて行列のデータを格納，CPUにデータが到着した後，単精度浮動小数点数を倍精度浮動小数点数に型変換し計算する，**BLASは利用しない**

単精度浮動小数点数:連続して表現可能な整数の範囲，-1.0×2^{24}から1.0×2^{24}

$$p \leq 2\sqrt{\frac{2^{53}}{N}+1} \quad \text{かつ} \quad p \leq 2^{25}+1$$

であれば，CPUとメモリの間は単精度浮動小数点数，CPUの中では，倍精度浮動小数点数を用いて計算可能

技法.3:整数のためのSIMD演算ユニットを使う方法

$0 < p \leq \sqrt{\frac{2^{64}-1}{N}} + 1$を満たす素数$p$を法とする有限体$\mathbb{Z}/p\mathbb{Z}$を考える,

以下のプログラムは, 整数のためのSIMD演算ユニットを有効に活用できる

```
typedef unsigned int UI;
typedef unsigned long long ULL;
UI a[N],b[N],r,p;
ULL t=0;
for(i=0;i<N;i++) t+=(ULL)a[i]*b[i];
r=t % p
```

技法.4:整数の剰余計算の高速化

$c_i = a_i \bmod p$を考える

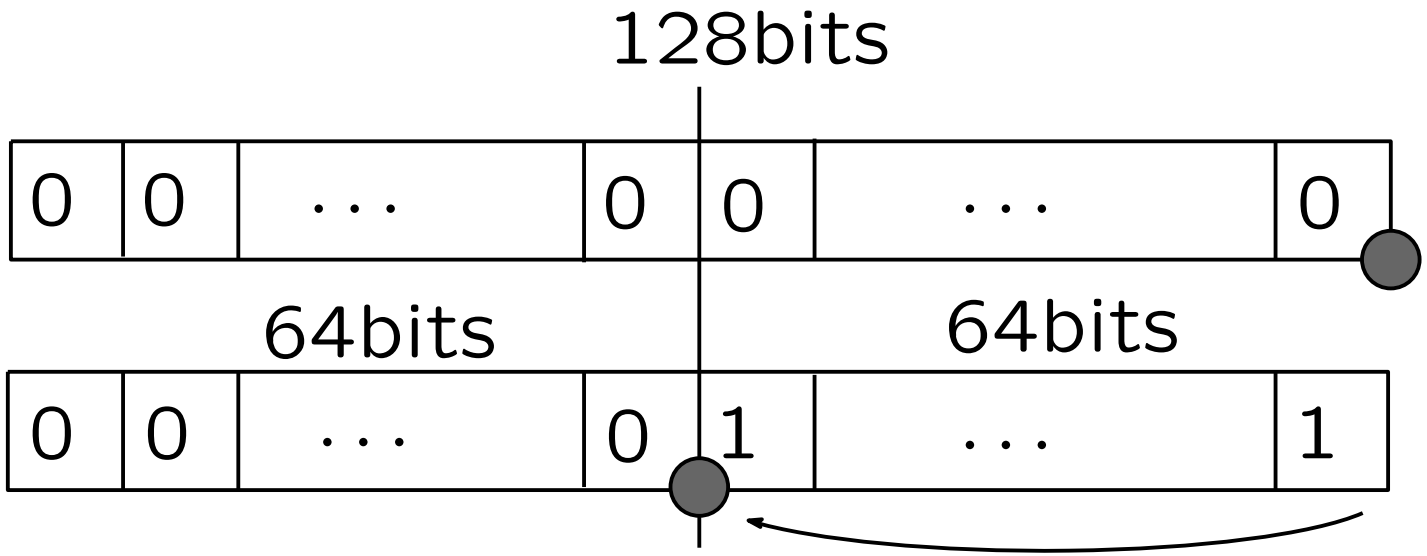
$0 \leq c_i < p$を要求されると，CPUの命令をそのまま呼び出す以外に方法はない

もし，c_iが，$0 \leq c_i < 2p$の範囲にあればよいという緩い条件の場合には，次のようにして演算を高速化することが可能である

$$M = \frac{2^{64} - 1}{p} \quad \text{(切り捨て演算)}$$

を用意する，M を，擬似逆元という

擬似逆元の意味を考える，原点を移動する



$0.1111111111111111111111111111111111111111111111111111111111111111_{(2)} \approx 1.0$

より，$M = (2^{64} - 1)/p \approx 1.0/p \times 2^{64}$

pが一定ならば，Mも一定

$(a_i \times M)$の結果の上位64bits

は，a_i/pの除算をした時の切り捨て演算における真の商に等しいか，または，切り捨て演算における真の商より1小さい(ただし，負の数にはならない)

$c_i = a_i - p \times (a_i \times M$の結果の上位64bits)

64bitsの数同士の乗算と計算結果の上位64bitsの取り出し

インラインアセンブリ言語，xxxは，ダミーの変数

```
__asm__ ("mulq %3":"=a"(xxx),"=d"(r):"a"(a1),"g"(a2))
```

Tropical Determinantの話

行列式の定義

A:n次正方行列$A = (a_{ij})$に対して，その行列式det Aを

$$\det A = |A| = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{n\sigma(n)}$$

により定義する

Permanentの定義

A:n次正方行列$A = (a_{ij})$に対して，その行列式per Aを

$$\mathrm{per}\, A = \sum_{\sigma \in S_n} a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{n\sigma(n)}$$

により定義する

ultradiscretization

以下の規則に従って，演算を変更すること

$$a + b \rightarrow \max(a, b)$$
$$a \times b \rightarrow a + b$$
$$a/b \rightarrow a - b$$

ultradiscrete permanentの定義

A:n次正方行列$A = (a_{ij})$に対して，その行列式udper Aを

$$\text{udper } A = \max_{\sigma \in S_n} a_{1\sigma(1)} + a_{2\sigma(2)} + \cdots + a_{n\sigma(n)}$$

により定義する

ultradiscrete permanentは，Tropical Determinantとも呼ばれている

Tropical Determinantの計算アルゴリズム

Linear Assignment Problem

最短路問題に帰着

Tomizawa, N. : On some techniques useful for the solution of transportation problems. Networks 1, 173 194 (1971).

Jonker-Volgenant algorithm(LAPJV), 1987

Tropical Determinantの数式処理ソフトへの応用

$$A = \begin{vmatrix} 2x^3+1 & x^3+6x^2+5 \\ 7x^5+11x^4+3 & 9x^3 \end{vmatrix}$$

より,

$$B = \begin{pmatrix} 3 & 3 \\ 5 & 3 \end{pmatrix}$$

を得る, BのTropical Determinantを計算することで, Aの計算結果の最大次数の上界が手に入る

Newton補間の話

Newton補間の実装法

1変数n次多項式$f(x)$のNewton補間

$$f(x) = f[x_0] + (x - x_0)f[x_0, x_1] + ... \\ +(x - x_0) \cdots (x - x_n)f[x_0, x_1, x_2, ..., x_n]$$

における係数$f[x_0, x_1, x_2, ..., x_n]$は，1階の差分商を，

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0},$$

2階の差分商を，

$$f[x_0, x_1, x_2] = \frac{f[x_0, x_2] - f[x_0, x_1]}{x_2 - x_1},$$

と定義し，n階の差分商を

$$f[x_0, x_1, ..., x_n] = \frac{f[x_0, x_1, ..., x_{n-2}, x_n] - f[x_0, x_1, ..., x_{n-2}, x_{n-1}]}{x_n - x_{n-1}},$$

と定義する.

$x_n - x_{n-1}$などを逆元として保持するテクニックは使える，しかし，$\mathbb{Z}/p\mathbb{Z}$では，桁溢れをおこす危険を回避するため，1回の差分商毎に，剰余演算を行う必要がある

Newton補間の正しい実装法(1)

$(x_0, f(x_0)), (x_1, f(x_1)), \cdots, (x_N, f(x_N))$が与えられているとき，すべての点を通る多項式

$$f(x) = f_0 + (x - x_0)f_1 + (x - x_0)(x - x_1)f_2 + \cdots + (x - x_0)(x - x_1)\cdots(x - x_N)f_N$$

を求めよ

$$f(x_0) = f_0, f(x_1) = f_0 + (x_1 - x_0)f_1$$

より,

$$f_1 = \frac{f(x_1) - f_0}{x_1 - x_0}$$

$$f(x_2) = f_0 + (x_2 - x_0)f_1 + (x_2 - x_0)(x_2 - x_1)f_2$$

より,

$$f_2 = \frac{f(x_2) - (f_0 + (x_2 - x_0)f_1)}{(x_2 - x_0)(x_2 - x_1)}$$

多項式の評価$f_0 + (x_2 - x_0)f_1$が, このアルゴリズムでは重要な役割を果たす

Newton補間の正しい実装法(2)

$$f(x_j) = f_0 + (x_j - x_0)f_1 + (x_j - x_0)(x_j - x_1)f_2 + \cdots + (x_j - x_0)(x_j - x_1)\cdots(x_j - x_N)f_N$$

$(x_j - x_0), (x_j - x_0)(x_j - x_1), \cdots, (x_j - x_0)(x_j - x_1)\cdots(x_j - x_N)$をあらかじめ計算しておくとすると，多項式の評価とは，

$$f(x_j) = 1 \times f_0 + \left\{(x_j - x_0)\right\} \times f_1 + \left\{(x_j - x_0)(x_j - x_1)\right\} \times f_2 + \cdots + \left\{(x_j - x_0)(x_j - x_1)\cdots(x_j - x_N)\right\} \times f_N$$

ベクトルの内積と思うことができる，**内積は，整数のSIMD演算ユニットを使って実装できる**

技術の検証

Xeon Phi 7120Aの性能

Xeon Phi 1.238GHz 61core

理論ピーク演算性能:1208 GFlops

Host CPU:Xeon E5-1620 v2 @ 3.70GHz 4 cores

理論ピーク演算性能:118.4 GFlops

System D(CRAY XC30)の1ノード

Xeon E5-2695 v3 2.3GHz 14core × 2

理論ピーク演算性能:1030.4 GFlops

one-stage algorithmによるXeon Phi 7120Aの性能評価

$N \times N$の整数行列の固有多項式の計算時間の比較

各要素は, [1,10]の整数

N	Xeon Phi	XC30 1node
1000	21.423sec	13.81sec
2000	3m43.609sec	3m41.18sec
3000	15m56.151sec	17m48.96sec

Xeon Phi:Intel Compiler 15.0.2, XC30:gcc 4.9.2

それぞれ最速のタイミングデータを出すコンパイラを採用

理化学研究所 中田真秀氏によるMPACK 0.8.0を利用

ベンチマーク問題

```
E6(a)=a^27+12*p2*a^25+60*p2^2*a^23-48*p1*a^22+(168*p2^3+96*q2)*a^21
-336*p2*p1*a^20+(294*p2^4+528*q2*p2+480*p0)*a^19+(-1008*p2^2*p1-1344*q1)*a^18
+(144*p1^2+336*p2^5+1152*q2*p2^2+2304*p0*p2)*a^17
+((-1680*p2^3-768*q2)*p1-5568*q1*p2)*a^16
+(608*p2*p1^2+252*p2^6+1200*q2*p2^3+4768*p0*p2^2+17280*q0-1248*q2^2)*a^15
+((-1680*p2^4-2688*q2*p2+2304*p0)*p1-8832*q1*p2^2)*a^14
+(976*p2^2*p1^2+3264*q1*p1+120*p2^7+480*q2*p2^4+5696*p0*p2^3+
(43776*q0-4800*q2^2)*p2+12288*q2*p0)*a^13
+(832*p1^3+(-1008*p2^5-3072*q2*p2^2+5888*p0*p2)*p1-6528*q1*p2^3
+10752*q2*q1)*a^12
+((704*p2^3+4224*q2)*p1^2+2688*q1*p2*p1+33*p2^8-144*q2*p2^5+4384*p0*p2^4
+(41472*q0-6720*q2^2)*p2^2+34560*q2*p0*p2-34560*p0^2)*a^11
+(2560*p2*p1^3+(-336*p2^6-768*q2*p2^3+3584*p0*p2^2+64512*q0+8448*q2^2)*p1
-2112*q1*p2^4+23040*q2*q1*p2-70656*p0*q1)*a^10
+((176*p2^4+8960*q2*p2-18944*p0)*p1^2-5504*q1*p2^2*p1+4*p2^9-192*q2*p2^6
+2176*p0*p2^5+(22528*q0-3840*q2^2)*p2^3+32768*q2*p0*p2^2-39936*p0^2*p2
+110592*q2*q0-40704*q1^2+5120*q2^3)*a^9
```

```
+(2688*p2^2*p1^3+4608*q1*p1^2+(-48*p2^7+768*q2*p2^4-1536*p0*p2^3
+(82944*q0+16128*q2^2)*p2-73728*q2*p0)*p1-192*q1*p2^5+13824*q2*q1*p2^2
-64512*p0*q1*p2)*a^8
+(-2560*p1^4+(-32*p2^5+5376*q2*p2^2-16384*p0*p2)*p1^2+(-6144*q1*p2^3
-15360*q2*q1)*p1-48*q2*p2^7+608*p0*p2^6+(9600*q0-480*q2^2)*p2^4
+10752*q2*p0*p2^3-20992*p0^2*p2^2+(156672*q2*q0-38400*q1^2+9984*q2^3)*p2
-165888*p0*q0-56832*q2^2*p0)*a^7
+((1024*p2^3-10240*q2)*p1^3+10240*q1*p2*p1^2+(384*q2*p2^5-1792*p0*p2^4
+(21504*q0+6912*q2^2)*p2^2-57344*q2*p0*p2+49152*p0^2)*p1+1536*q2*q1*p2^3
-19456*p0*q1*p2^2-110592*q1*q0-21504*q2^2*q1)*a^6
+(-1536*p2*p1^4+(-16*p2^6+768*q2*p2^3-4608*p0*p2^2+27648*q0-19200*q2^2)*p1^2
+(-1344*q1*p2^4+10752*q2*q1*p2-9216*p0*q1)*p1+64*p0*p2^7
+(2304*q0+192*q2^2)*p2^5-3072*p0^2*p2^3+(55296*q2*q0-12288*q1^2
+4608*q2^3)*p2^2+(-110592*p0*q0-46080*q2^2*p0)*p2+73728*q2*p0^2)*a^5
+((64*p2^4-4096*q2*p2+8192*p0)*p1^3-512*q1*p2^2*p1^2+(-256*p0*p2^5+
(3072*q0-768*q2^2)*p2^3-8192*q2*p0*p2^2+16384*p0^2*p2+73728*q2*q0
-39936*q1^2-18432*q2^3)*p1-1024*p0*q1*p2^3+(-36864*q1*q0-3072*q2^2*q1)*p2
+24576*q2*p0*q1)*a^4
+(256*p2^2*p1^4+15360*q1*p1^3+(128*q2*p2^4-1024*p0*p2^3+(-6144*q0
```

```
-2560*q2^2)*p2+8192*q2*p0)*p1^2+(-128*q1*p2^5+2048*q2*q1*p2^2
-14336*p0*q1*p2)*p1+256*q0*p2^6-256*q2*p0*p2^5+256*p0^2*p2^4+(9216*q2*q0
-2560*q1^2-256*q2^3)*p2^3+(-18432*p0*q0-7680*q2^2*p0)*p2^2
+24576*q2*p0^2*p2-110592*q0^2+55296*q2^2*q0-30720*q2*q1^2-16384*p0^3
-6912*q2^4)*a^3
+(-1024*p1^5+4096*p0*p2*p1^3+24576*q2*q1*p1^2-12288*q1^2*p2*p1)*a^2
+(-2048*q2*p1^4+2048*q1*p2*p1^3+((-3072*q0-256*q2^2)*p2^2+4096*q2*p0*p2
-4096*p0^2)*p1^2+(512*q2*q1*p2^3-1024*p0*q1*p2^2-36864*q1*q0
+9216*q2^2*q1)*p1-256*q1^2*p2^4-6144*q2*q1^2*p2+12288*p0*q1^2)*a
+(4096*q0-1024*q2^2)*p1^3+(2048*q2*q1*p2-4096*p0*q1)*p1^2
-1024*q1^2*p2^2*p1-4096*q1^3
```

$E6(a)$の判別式を計算する

$E6(a)$の判別式計算

CPU:Intel Core i7 980X(6Core)

Mem:24G

Compiler:GCC 4.8.0

Option:-O3 -mtune=native -march=native -fopenmp

計算結果の項数:27329463項(txt形式:2.5G)

Serial: 10913m45.857s

Parallel: 1773m28.272s

Speed Up: 6.15