2004年の「スーパーコンピュータ・コンテスト」で高２の小野嶋勇介君と林信吾君のチームが優勝しました。そこで、「論集編集委員会」として、この大会は、どのような問題が出題され、どのように解いたのかを、後輩や仲間に紹介してほしいと、原稿を依頼しました。小野嶋君が心よく引き受けてくれ、臨場感あふれるものになっていますので、ご一読下さい。

（論集編集委員会）

# 第10回スーパーコンピュータコンテスト優勝記

高２―３　小野嶋　勇介

## １．スパコンとは何か

スパコンとは、毎年東京工業大学で行われている、スーパーコンピュータを使って高校生にさまざまな問題を解かせるプログラミングコンテストのことである。

今回は第10回記念大会ということで、国内のみではなくアジア４ヶ国（中国、韓国、タイ、シンガポール）からも参加。一応麻布は「常連校」ということになっていた。

## ２．予選問題＆解法

今年の予選問題は次のようなものであった。

【問題概要】

素因数分解した時（重複する分も含め）、素数の延べ個数がちょうど12個となる整数をSuperCon数と定義します。

例）3750000＝2＾4*3*5＾7（素数２が４個、３が１個、５が７個の合計12個）

与えられた整数n（ただし10,000,000≦n≦20,000,000）に対し、nから始めて2004番目のSuperCon数を求めるプログラムを作成しなさい。ただし、与えられた整数nそのものがSuperCon数の時はnを１番目のSuperCon数とみなします。

毎年の事ながらおかしな数学問題であるが、去年の予選問題のように図形的考察が必要無い分、若干楽であるように思えた。以下はこの問題を解くにあたってのアイディア集。

**・素数の組み合わせ**

SuperCon数は12個の素数から出来ているため、nから１ずつ足してそれがSuperCon数かどうかをチェックしていくより、12個の素数を掛け合わせてSuperCon数を作り、それが何番目のSuperCon数なのかを調べたほうが、効率が良い。これは少し考えれば誰でも分かる事なので、どのチームも使っていたと思う。

**・一気にまとめて算出**

１個目のSuperCon数を探し、次に２個目、３個目……というように、2004個目まで１つずつ求めていくのでは効率が悪い。必要なのは2004番目のSuperCon数だけなのだから、途中のSuperCon数の値や順番はどうでも良いのである。そこで、一定範囲内のSuperCon数を全部まとめて求め、2004個に達していなかったら範囲をずらして同様の処理を行い、2004個に達していたら求めたSuperCon数を昇順に並び替え、2004個目を取り出す、という手法を取った。

**・素数＆逆素数表**

今回の問題では、その性質上計算に素数を多用する。よって、素数をいつでも使えるように、プログラムの先頭で十分な大きさの素数表を作っている。

また、一定範囲内のSuperCon数を求める際、「その数は何番目の素数かor何番目と何番目の素数の間に存在する数か」という情報が繰り返し必要となる。そのたびに素数表から求めることも不可能ではないが、かなりの回数必要となるため、いちいち計算するのは、それなりの負担となる。そこで、自然数から何番目の素数かを一発で求めることができる表を、プログラムの先頭で作成した。この表のことを、我々は「逆素数表」と呼んでいた。

あとはこれらのアイディアを元に、ひたすらプログラミング。一週間ほどで１ミリ秒台（図書館棟PC）で動くようになったため、高速化はもういいだろうということでプログラムはひとまず完成。残りの期間で解法を説明するためのレポートの作成に取り掛かった。予選ではこのレポートも評価対象となるので、手を抜くことは出来ない。

なんとか締め切り当日の早朝にレポートを仕上げ、東工大へメール転送。数週間後、予選通過を知らされ、とりあえず一息。

## ３．本選問題

本選問題は、以下の通り。

【スーパーコン04の問題】

暗号化された画像のファイルeimagefile、ヒント情報のファイルhintfileに与えられるデータをもとに、できる限り正確な元画像を求めるプログラムを作成せよ。

プログラムの性能は、評価用に用意された１組のファイルに対し、制限時間（３分）以内に、出力された画像の正解率（元画像と一致する画素数の割合）で評価する。ただし、同じ正解率の画像を出力した場合には、出力までの計算時間が短

いものを良いプログラムとする。
（注：制限時間内であれば、何度でも画像を出力してよい。複数出力された場合には、最後に出力された画像を用いる。）

この問題を簡単に説明すると、まず元画像は暗号化された画像のファイルと「各画素の位置に対応した数値」の2つから単純な演算で求めることが出来る。暗号化された画像のファイルは初めから与えられているので、あとは「各画素の位置に対応した数値」を求めれば良いわけだが、これは「画素の位置」を、ある関数に通せばすぐに求めることが出来る。しかし、この関数の中身は、スパコン開催者側しか知らない秘密のもので、関数の中身を求めるのは難しい。

それでは、どうやって「各画素の位置に対応した数値」を求めるかというと、ヒント情報を駆使することにより、多少時間はかかるが求めることが出来る。ただし、画素によって、ヒント情報に「困難さ」が設定されており、「困難さ」の値が高ければ高いほど「各画素の位置に対応した数値」を求めるのに時間がかかる。

これらの方法を使って、より多くの画素を正確に求められるかが、今回の問題だ。

メンバーは以下の二人

麻布学園高二　小野嶋　勇介
林　信吾

ちなみに、「nemui」＝「眠い」というカッコよさのかけらもない我々のチーム名は、予選のレポートを夜遅くまで書いていた時に決めたもの。その時の心理状態をそっくりそのまま反映させた非常に適当なネーミングである。もう少しまともな名前も考えたのだが、どれもいまいちということで、最終的にこれになってしまった。

## 4．いざ東工大へ

コンテスト期間は、8月2日～7日までで、東工大本校のある大岡山で行われた。

各出場チームについては、以下の通り。

| チーム | 学校名・国名（県名） | メンバー | | |
|---|---|---|---|---|
| rousers | 一関工業高等専門学校（岩手） | 熊谷　一生 | 佐藤　　光 | 中村　健大 |
| ekomix | 早稲田高等学校（東京） | 林　　吟志 | 上野　裕介 | 古谷　陽平 |
| nemui | 麻布高等学校（東京） | 小野嶋勇介 | 林　　信吾 | |
| kstpcc | 川崎市立川崎総合科学高等学校（神奈川） | 村松　雄介 | 久保　知弥 | |
| nomean | 浅野高等学校（神奈川） | 宮崎　晃一 | 藤井　理史 | 松永　賢太 |
| iacc | 石川県立金沢泉丘高等学校（石川） | 尾崎　順一 | 斉藤　　学 | 中村　翔太 |
| nagater | 兵庫県立長田高等学校（兵庫） | 釜江　典裕 | 寺崎　大洋 | 平澤　恭治 |
| scorpion | 灘高等学校（兵庫） | 舟山　和男 | 中川　雄大 | |
| emisi | 灘高等学校（兵庫） | 老木　智章 | 内藤　皓太 | |
| ydobon | 八代工業高等専門学校（熊本） | 錦戸　裕輔 | 岩上　拓矢 | 中村　一貴 |
| threein1 | シンガポール | Tam Wai Lun | Liang Junjie | Sim Yan Chuan |
| bsabsa | 韓国 | Sungwoo Choo | Daegun Won | Sungkwang Lee |
| china | 中国 | Tang yue | Li Hao | Zhu Zeyuan |
| istar | タイ | Sakolrat Titaram | Pathompol Seang-Uraiporn | Chawawat Chansiricharoengul |

8月2日。

我々は東工大に集まり、UNIXやX端末の使い方、スーパーコンピュータの仕組みやセキュリティーカードの使い方についての説明を受け、いよいよスーパーコンピュータ本体の見学へ。

……一言で言うと「やたらでかい」。人の身長ほどのコンピュータが詰まったボックスが大量に並んでおり、部屋全体の床下からすごい勢いで吹き上げる空調、身長ほどの高さまで綺麗に積み上げられたハードディスクのタワーなど、とにかく一般のパソコンとは比べ物にならないほどのスケールだ。

ちなみに、全部合わせると60億円もするらしい。うーん高い。

その後、恵比寿にある日本シリコングラフィックス社（SGI）を見学。この会社は、今回我々が使用したスーパーコンピュータを東工大に納入した会社である。ここで、最新のコンピュータグラフィックスの世界を体験し、その後帰宅。

## 5．本選問題解法

次の日から、早速本選問題の解答プログラム作成を開始。以下はそのアイディア集。

**・困難さ順にソート**

ヒント情報の「困難さ」が1増えると、画素を求めるのに約2倍も時間がかかる。なるべく多くの画素を求めるため、困難さが低い画素から順に解いていくことにした。

**・積の法則**

3分という限られた時間では、いくらスパコンといえども、ヒントから単純に計算するだけで全ての画素を求めようとしたら全体の1割程度の画素しか求めることが出来ない。そこで、今回の問題の次のような性質を利用することにした。

位置a、bの画素が求まったら、位置a*bの画素も自動的に求まる

例えば、3と4の位置の画素が求まったら、3*4＝12の位置の画素も自動的に求まるのである。さらにそこから12*3＝36の位置、12*4＝48の位置……と、再帰的にこの法則を適用すれば、次々と画素が求まる仕組みだ。これを利用すれば、「困難さ」がいくら高い画素でも、一瞬で求めることが出来る。

なお、この法則は配られた問題用紙の最後にヒントとして載っていたため、多くのチームが利用している。

**・除の法則**

かなり頼れる武器である「積の法則」だが、これには大きな弱点が存在した。

説明したとおり、積の法則では、すでに求めた画素の位置を掛け合わせた場所の画素が求まる。そのため、すでに求めた画素の位置があまり大きな数では、掛け合わせると画素が画像からはみだした位置になってしまい、使い物にならない。しかも、後の方にある画素ほど「困難さ」が低く、逆に前の方にあるほとんどの画素は「困難さ」が高い。これでは積の法則で求められる画素などたかが知れている。

あれこれ悩んでいるうちに、次のような事を思いついた。「積の法則が存在するのなら、その逆の『除（割り算）の法則』があってもいいんじゃないか？」

つまり、「位置a、a*bの画素が求まったら、位置bの画素も自動的に求まる」という法則が成り立つのではないか、ということだった。これがもし本当に成り立てば、後の方にある画素から、前の方の画素を求めることが出来る。

計算してみたところ、実際に使えることが判明したので、バグに悩まされながらも何とかプログラムに組み込んだ。

**・並列処理**

スーパーコンピュータ・コンテストでの一番の醍醐味は、この並列処理である。一般のパソコンには、普通CPU（中央演算装置）は1個しか付いていないものだが、スーパーコンピュータには、それが数十、数百個も付いている。スーパーコンピュータが速いと言われている理由はそこにある。

しかし、普通にプログラムを作っても、CPUは1個しか使われない。CPU1個では、一般のパソコンの性能と大差ないので、これでは意味がない。CPUを複数個使って計算させるには、プログラムを「並列化」しなくてはならない。

一言で並列化と言っても、様々な方法が考えられる。どの部分を、どのように並列化するかが問題になってくるが、我々のプログラムでは、動作を極力簡潔にするため

（並列処理は複雑に設計すればするほど作るのが難しく、バグが出る可能性も高くなる）マスター&スレーブ方式（1つのCPUが、残りのCPUの演算を制御する）を採用し、マスターでデータの管理・スレーブとのデータのやり取り・積&除の法則の計算を行い、スレーブでは各画素を「困難さ」の低い順に計算させることにした。

この並列化で、新たな問題が発生した。本来、使用するCPU数と処理速度はほぼ比例するはずなのだが、使用するCPU数を増やしても、ほとんど速度が変わらないのだ。

原因はマスターの計算量にあった。マスターは、新しく画素が求まるたびに積の法則、除の法則を計算しており、その間スレーブとのデータのやり取りは保留される。そのため、マスターがいわばビジー状態となり、スレーブとのデータのやり取りがスムーズに行われず、結果スレーブに「計算をしない時間」が多く生じてしまうのだ。

この問題を解決するために、マスターが積の法則、除の法則の計算を行う回数を10分の1、つまり新しい画素が10個求まるたびに1回計算する、という手段を取り、マスターの負担を軽減した。法則の適用のタイミングに多少ムラが出てしまうが、スレーブとのデータのやり取りが停滞することよりはだいぶマシである。

**・定期出力**

今回の大会では、「より早く、より多くの」画素を求めたチームが勝利するわけだが、3分以内であれば、早さよりも画素の多さの方が優先して評価される。つまり制限時間を目一杯使って、一つでも多くの画素を求めたほうが良いのである。そのため、我々のプログラムでは5秒に1回の周期で、結果を出力するようにした（最後の出力が評価対象となる）。

**・最適化**

画素の計算、積&除の法則、並列処理……今まで書いてきたソースを全て見直し、ギリギリまで高速化を施した。特に、画素を求める際の計算は、膨大な回数行うので、念入りに高速化を行った。

余談だが、私はこの最適化という作業が一番好きである。小一時間ソースと向かい合い、書いては消し、書いては消しを延々と繰り返して、ミリ秒単位で速度を上げていく楽しさは、ある程度プログラミングを経験している者なら分かると思う。

……分からない？そいつは残念。

**・補間法**

この方法は、決してセオリーとは言えないのだが、導入すると最終的な精度が8～9割上がるため、実装することにしたものである。

補間法とは、まだ求まっていない画素を、すでに求めてある周りの画素の色から予想する、言わばギャンブル的な方法である。この方法は、「答えとなる画像は写真か何かである」という根拠のない前提条件の上ではじめて成り立つ。つまり、答えの画像に色のムラがあることを利用しており、当然答えの画像がランダムノイズだったりしたら、この方法は全く役に立たない。

## 6．結果発表

締め切りギリギリまで調整、テストランを行い、不具合がないことを確認してプログラムを提出。4日間に及ぶプログラミング漬けの生活に幕を閉じた。

そして、結果発表の日。

各チームごとに、プログラムの説明を簡潔に行い、続いてサンプルデータ（本番用のデータとは別のもので、ヒントが若干優しくなっている）でプログラムを走らせた時の実行結果が表示された。

我々の発表の番となり、緊張しながらプログラムの説明を終え、続いてサンプルデータでの実行結果が表示された。我々のプログラムは若干の誤差を出したものの、かなり多くの画素を正確に求めていた。が、中国チームはこのサンプルデータで100%の正答率をマークする。

結局サンプルデータでは、中国チームが他のチームを引き離し、1位となった。

そして、いよいよ本番用データでプログラムを走らせた時の結果発表。3位から順に発表されていく形式であった。

第3位　チーム「nomean」82.60%。浅野高等学校が正答率82%で第3位となった。

第2位　チーム「china」99.81%。サンプルデータでは1位だった中国チームが、本番では2位という結果。ということはもしや……。期待と不安が同時に広がった。

そして第1位の発表。

第1位……チーム「nemui」99.92%。やった！

中国チームとはわずか0.11%差。画素数で言えばたった71個差という大接戦であった。後で知ったのだが、制限時間のラスト5秒というところで、我々のプログラムは一気に大量の画素を求め、最後の最後で中国チームを抜いたらしい。

麻布チームとしては実に7年ぶり2度目の優勝。「いつかこの大会で優勝してやる」という夢が叶った瞬間だった。

「東工大ホームページアドレス　http://www.gsic.titech.ac.jp/」

付録・チーム「nemui」予選解答プログラム

```
/*************************************************************************/
/*                                                                       */
/*      スーパーコンピュータコンテスト 2004 予選問題解答                 */
/*                                                                       */
/*      Program by : Team NeMuI                          NeMuI.c         */
/*                                                                       */
/*************************************************************************/
#include <stdio.h>
#include <stdlib.h>

#define RANGE       500000      /* 一度に求める SuperCon 数の範囲 */
#define PRIME       20000       /* 素数&逆素数テーブルのサイズ */
#define SUPERCON    3000        /* SuperCon 数テーブルのサイズ  */
#define LAYER       13          /* 階層数(12 層+始点)           */
#define HIGHEST     1           /* 最上層の位置                 */
#define LOWEST      (LAYER-1)   /* 最下層の位置                 */
#define NUM         2004        /* 何番目の SuperCon 数を求めるか */

int prime_table[PRIME];         /* 素数テーブル         */
int r_prime_table[PRIME];       /* 逆素数テーブル       */
int supercon_table[SUPERCON];   /* SuperCon 数テーブル  */

/*************************************************************************/
/*                          qsort 用比較関数                             */
/*************************************************************************/
int compare(const int *arg1, const int *arg2)
{
  if(*arg1 > *arg2)
    return 1;
  else if(*arg1 < *arg2)
    return -1;
  return 0;
}
```

```
/*************************************************************************/
/*                    素数＆逆素数テーブル作成関数                       */
/*************************************************************************/
void prime(void)
{
  int i,j;
  int prime_count = 0;    /* 求めた素数の個数 */

  for(i=0; i<PRIME; i++)  /* 素数テーブルをフラグテーブルとして初期化 */
    prime_table[i] = 1;

  for(i=0; i<PRIME; i++)  /* 素数＆逆素数テーブルの作成 */
  {
    r_prime_table[i] = prime_count;
    if(prime_table[i]==1 && i>=2)
    {
      prime_table[prime_count] = i;
      prime_count++;
      for(j=i*2; j<PRIME; j+=i)
        prime_table[j] = 0;
    }
  }
}

/*************************************************************************/
/*            引数 n から 2004 番目の SuperCon 数を返す関数              */
/*************************************************************************/
int supercon(int n)
{
  int i, j;
  int max, min;           /* 一度に求める SuperCon 数の上限＆下限 */
  int layer;              /* 現在の階層                           */
  int supercon_count = 0; /* 求めた SuperCon 数の個数             */
  int old_supercon_count; /* 前回求めた SuperCon 数の個数         */
  int factor[LAYER];      /* 各層ごとの素数番号                   */
  int product[LAYER];     /* 各層までの素数の積                   */
```

```
min = n;
max = n+RANGE;
product[HIGHEST-1] = 1;
while(supercon_count < NUM) /* SuperCon 数が 2004 個以上見つかるまで繰り返す */
{
  old_supercon_count = supercon_count;
  layer = HIGHEST;
  factor[HIGHEST] = 0;
  while(layer >= HIGHEST) /* 範囲内の SuperCon 数を全部求めるまで繰り返す */
  {
    for(i=layer; i<=LOWEST; i++)    /* 現在の階層から最下層まで初期化 */
    {
      factor[i] = factor[layer];
      product[i] = product[i-1]*prime_table[factor[i]];
    }
    if(product[LOWEST] < max) /* 最下層までの積が上限を超えていないかチェック */
    {
      if(product[LOWEST] < min)
        i = r_prime_table[(min-1)/product[LOWEST-1]+1];
      else
        i = factor[LOWEST];
      j = r_prime_table[(max-1)/product[LOWEST-1]+1];
      while(i < j)    /* SuperCon 数を求めてテーブルに格納 */
      {
        supercon_table[supercon_count] = product[LOWEST-1]*prime_table[i];
        supercon_count++;
        i++;
      }
      layer = LOWEST;
    }
    layer--;
    factor[layer]++;
  }
  min = max;
  max += RANGE;
}
```

```
  qsort(supercon_table+old_supercon_count, supercon_count-old_supercon_count,
                        sizeof(int), (int(*)(const void*,const void*))compare);

  return supercon_table[NUM-1];
}

/*************************************************************************/
/*                              メイン関数                               */
/*************************************************************************/
int main(void)
{
  int n;

  scanf("%d",&n);
  prime();
  n = supercon(n);
  printf("%d", n);

  return EXIT_SUCCESS;
}
```

汚いプログラムですが、図書館棟のコンピュータスペースなど、C言語のコンパイラがインストールされているパソコンで実行すれば、一応動きます。

本選プログラムはホームページアドレス「http://www.gsic.titech.ac.jp/supercon/supercon2004/kaisetsu/nemui.c」にあります。予選プログラムより何倍も汚いプログラムですが、暇な方はどうぞ。