T2K オープンスパコンを用いた３次元津波伝搬シミュレーションにおけるコードチューニング一事例

片桐　孝洋

東京大学情報基盤センター　特任准教授

１．はじめに

地震により生じる津波の予測は防災上きわめて重要である。現在、スーパーコンピュータを利用し、地震波と連動して起こる津波伝搬シミュレーションの手法が研究されている[1-3]。

従来の津波伝搬シミュレーションでは、長波近似に基づく2次元空間における津波伝搬シミュレーションが主流であった。しかし、このような近似モデルが入ることにより津波を過大に予測してしまうことから、近似を用いることなく3次元空間でのシミュレーション手法の開発が期待されており、すでに手法が提案されている[4]。3次元空間での津波伝搬シミュレーションは主要カーネルに対し O(nx * ny * nz)の計算量が必要となり、2次元空間での O(nx * ny)に比べ計算量が増大なことから高速化が必須である。

本稿では、機械語の利用やコンパイラ最適化オプションの調整ではない、コードレベルの書き換えによる高速化を目指す。本稿で示される最適化の一部は、コンパイラオプションの指定により達成可能であるが、コンパイラ最適化の仕組みを理解いただくことと、コンパイラ性能に依存しない最適化手法（方法論）を習得することが重要であるので、チューニング一例として記事で紹介する。

本稿の構成は以下のとおりである。2節において本稿で取り扱う3次元津波伝搬シミュレーションコードの概略を説明する。本稿では主要ループ（演算カーネル）に注目し、考えられるコードの最適化手法について説明する。3節で実際計算機である。T2K オープンスパコン（東大）を利用し、2節のコードの性能評価を行う。最後に、本稿で得られた知見を述べる。

２．３次元津波伝搬シミュレーションコード

（１）概要

3次元の津波生成と伝搬をシミュレーションするための Navier-Stokes 方程式を有限差分法で離散化したものである。自由表面をもつ流体の解析手法である SOLA-SURF 法[5]をもとにしている。数値計算アルゴリズム上の分類は、有限差分法の陽解法となる。求めるべき変数（3次元配列となる）は、x、y、z 方向に対応する速度 u、v、w と、圧力 p である。

（２）ホットスポット部分と演算カーネル

このコードの最も時間のかかる部分（ホットスポット）は、図 1 のようなループ構造となっている。

```
  do itl = 1, itlmax
     err = 0.0
     演算カーネル部分（err の計算）
     if (err. lt. eps) goto OUT
   enddo
OUT  continue
```

図 1 ホットスポット部分の構成

すなわち、要求精度 eps が満たされるか、最大反復回数 itmax まで、演算カーネル部分が呼ばれる。演算カーネルは、ｘ、ｙ、ｚ軸に関する３重ループとなる。それを図 2 に示す。

```
        do k = 1, nz
          do j = 1, ny
            do i = 1, nx
              if (mos(i,j,k)) then
                dd = (u(i,j,k)-u(i-1,j,k))*dxinv
&                  + (v(i,j,k)-v(i,j-1,k))*dyinv
&                  + (w(i,j,k)-w(i,j,k-1))*dzinv
                dp = beta*dd
                u(i,j,k)   = u(i,j,k)   + dtdx*dp
                u(i-1,j,k) = u(i-1,j,k) - dtdx*dp
                v(i,j,k)   = v(i,j,k)   + dtdy*dp
                v(i,j-1,k) = v(i,j-1,k) - dtdy*dp
                w(i,j,k)   = w(i,j,k)   + dtdz*dp
                w(i,j,k-1) = w(i,j,k-1) - dtdz*dp
                p(i,j,k)   = p(i,j,k) + dp
                err = max (err, abs(dd))
              endif
            enddo
          enddo
        enddo
```

基本
演算
コード

図 2　演算カーネル部分（論理マスクコード）

図 2 では、3 重ループの中央に演算を行うかどうか決定する論理マスク mos(i,j,k)が存在する。海面に相当する 2 次元座標（x, y）のループ変数は i, j ループとなるが、これらの変数は、座標(x, y)が海面である場合のみ計算することになる（つまり、陸地に津波はない）。もし、座標(x, y)が陸地の場合、ｚ方向、すなわちｋループ長が 0 であり、無駄な演算を行わないために論理マスク配列 mos () を導入している。

コード最適化の観点では、図 2 のコードの問題点は以下である。

I.　ＩＦ文がループの中央にあるため比較回数が多い。ＩＦ文により分岐予測が困難となり命令発行が先行して行えない。コード最適化の妨げとなる。

II. i，jループの値に依存してkループ長が決まるため、kループ長が一定でない。このことにより、k ループが連続となる仮定でのコード最適化（たとえば、データのプリロード）が実装できない。

上記問題Ｉを解決するには、i,j に依存する k ループ長を記憶した変数を導入すると、ＩＦ文が消去できる。以下の図 3 にコードを載せる。

```
do j = 1, ny
  do i = 1, nx
    kb2 = kb(i,j)+1
    kt2 = kt(i,j)-1
    if(kt2 .gt. kb2 ) then
      do k = kb2,kt2
        図 2の基本演算コード
      enddo
    endif
  enddo
enddo
```

図 3　ＩＦ文除去コード

図 3のコードでは、最内ループ中からＩＦ文が除去されているが、最内ループがkループになる。この問題点は、u，v，w，p 配列において、連続してデータが入っている方向がFortranではiループの方向であるため、これらの配列すべてにおいて連続アクセスにならない。このことは、キャッシュ上データの利用の妨げとなり、激しい性能劣化を引き起こす要因となる。

以上から、3 次元津波伝搬シミュレーションコードにおけるコード最適化は、単純な 3 重ループ（つまり、i，j，kループの範囲がすべて自明）でなく、コード最適化は容易ではない。

（3）高速化に向けたアルゴリズムの改良

ところで、図 2において強制的にＩＦ文を取り除いた場合を考えると、kループが連続となりデータアクセスと最適化の観点で好ましい。当然このコードは、すべてが均質的な深さの海になり、実際の地形情報を利用した３次元津波伝搬シミュレーションが達成できるものではない。かつ、地形データを利用した演算量削減が利用できないのと、演算量が増すという問題がある。しかし、ベンチマークコードとして性能が興味深いといえる。

```
do k = 1, nz
  do j = 1, ny
    do i = 1, nx
       図 2の基本演算コード
    enddo
  enddo
enddo
```

図 4　単純k,j,iループコード

k ループ長は座標(x,y)の地点における海の深さと同値である。津波伝搬シミュレーションの特性から、陸地の部分ではｋループ長は０もしくは極めて短い値となるが、一方で、深海の部分は一定区間で深いｋループ長をもつはずである。すなわち、地形情報の問題特性を利用すると、ある kblk*kblk の区間［x:x+kblk, y:y+kblk]で深海が存在すれば、ｋループ長が長くとれる箇所がある。この特徴を考慮したコードは、以下のようになる。

```
do jjj = 1, ny/kblk
  jj = 1 + (jjj-1)*kblk
  do iii = 1, nx/kblk
    ii = 1 + (iii-1)*kblk
    do k = kbmin(iii,jjj), ktmax(iii,jjj)
      do j = jj, jj+kblk-1
        do i = ii, ii+kblk-1
           図 2の基本演算コード
        enddo
      enddo
    enddo
  enddo
enddo
```

図 5　単純ブロック化コード

ここで、kbmin(iii,jjj)と ktmax(iii,jjj)は、区間［iii:iii+kblk, jjj:jjj+kblk]内すべての座標において保障されている､連続したｋループの開始値と終了値を記憶した配列である。

（４）一般化したブロック化コード

図 5 のブロック化コードは、kblk*kblk の区間内で、k-ループの連続した開始値と終了値が一定でないと使えない。実際の地形データは凹凸があるので、各区間において、連続した幅が取れる場合もあるし、取れない場合もある。取れる場合においても、ある範囲は一定した間隔がとれるが、一定した間隔で演算をしたあと、残りの部分を計算しないと演算結果が一致しない場合もある。

以上を考慮すると、図 5 のコードは、ｋループについて、1)ブロック化部分を計算する前の部分、2) ブロック化部分、3) ブロック化部分を計算した後の部分、の 3 部分に分割する必要がある。図 6 にそのコードを示す。

```
c          ====  １）ブロック化部分を計算する前の部分
           do jjj = 1, ny/kblk
             jj = 1 + (jjj-1)*kblk
             do iii = 1, nx/kblk
               ii = 1 + (iii-1)*kblk
```

```
          if (kbmin(iii,jjj) < ktmax(iii,jjj)) then
            do k = 1, kbmin(iii,jjj)-1
              do j = jj, jj+kblk-1
                do i = ii, ii+kblk-1
                  if (mos(i,j,k)) then
                    図 2の基本演算コード
                  endif
                enddo
              enddo
            enddo
          endif
        enddo
      enddo
c     ==== 2）ブロック化部分
      do jjj = 1, ny/kblk
        jj = 1 + (jjj-1)*kblk
        do iii = 1, nx/kblk
          ii = 1 + (iii-1)*kblk
          do k = kbmin(iii,jjj), ktmax(iii,jjj)
            do j = jj, jj+kblk-1
              do i = ii, ii+kblk-1
                図 2の基本演算コード
              enddo
            enddo
          enddo
        enddo
      enddo
c     ==== 3）ブロック化部分を計算した後の部分
      do jjj = 1, ny/kblk
        jj = 1 + (jjj-1)*kblk
        do iii = 1, nx/kblk
          ii = 1 + (iii-1)*kblk
          if (ktmax(iii,jjj) < kbmin(iii,jjj)) then
            kstart = 1
          else
            kstart = ktmax(iii,jjj)+1
          endif
          do k = kstart, nz
            do j = jj, jj+kblk-1
              do i = ii, ii+kblk-1
```

```
                        if (mos(i,j,k)) then
                          図 2の基本演算コード
                        endif
                      enddo
                    enddo
                  enddo
                enddo
              enddo
```

図 6　一般化ブロック化コード

図 6の2）ブロック部分以外における<図 2の基本演算コード>部分を含むループは、論理マスクを使ったものでも、ＩＦ文除去版でも利用できる点に注意する。なお、各 kblk*kblk の区間に連続した部分が存在しない場合は、3）部分でブロック化なしのコードが実行される。したがって図 6の性能は、最悪でもブロック化なしのコードと同一になる。

（5）その他の高速化可能性

- 不連続配列の局所化

いままでのコードでは、i ループ方向の連続性を考慮しているものの、j および k ループの方向のアクセスの際には、データがとびとびになり、キャッシュに悪影響を及ぼす。特に k ループ方向については nx*ny 間隔で離れており、j ループ方向のアクセスの nx 間隔に対してきわめて距離がある。そこで、k-1 のアクセスがある w 配列について、局所化された配列 w2 を用意し、その局所化された配列に元の配列 w をコピーしてから演算し、w2 の演算結果を再び w に書き戻す方法がキャッシュの観点で有効となりうる。この最適化を行ったコードを以下に示す。

```
              do jjj = 1, ny/kblk
                jj = 1 + (jjj-1)*kblk
                do iii = 1, nx/kblk
                  ii = 1 + (iii-1)*kblk
                  do k = kbmin(iii,jjj), ktmax(iii,jjj)
c                   --- 配列 w を局所配列 w2 にコピー
                    do j = jj, jj+kblk-1
                      do i = ii, ii+kblk-1
                        w2(i-ii+1,j-jj+1) = w(i,j,k-1)
                      enddo
                    enddo
                    do j = jj, jj+kblk-1
                      do i = ii, ii+kblk-1
                        dd = (u(i,j,k)-u(i-1,j,k))*dxinv
     &                     + (v(i,j,k)-v(i,j-1,k))*dyinv
     &                     + (w(i,j,k)
     &                         -w2(i-ii+1,j-jj+1))*dzinv
```

```
                dp = beta*dd
                u(i,j,k)   = u(i,j,k)   + dtdx*dp
                u(i-1,j,k) = u(i-1,j,k) - dtdx*dp
                v(i,j,k)   = v(i,j,k)   + dtdy*dp
                v(i,j-1,k) = v(i,j-1,k) - dtdy*dp
                w(i,j,k)   = w(i,j,k)   + dtdz*dp
c               --- 局所化された配列 w2 を用いて演算
                w2(i-ii+1,j-jj+1) =
     &              w2(i-ii+1,j-jj+1) - dtdz*dp
                p(i,j,k)   = p(i,j,k)   + dp
                err = max (err, abs(dd))
              enddo
            enddo
c           --- 局所配列 w2 を帯域配列 w に保存
            do j = jj, jj+kblk-1
              do i = ii, ii+kblk-1
                w(i,j,k-1) = w2(i-ii+1,j-jj+1)
              enddo
            enddo
          enddo
        enddo
      enddo
```

図 7　w 配列局所化コード

図 7 のコードは、コピー時間、書き戻し時間、および局所化された配列 w2 のアクセス時間が、元の配列 w のアクセス時間より短縮される場合に高速となる。

- 出力依存の排除

図 2 の基本演算コードは、配列 u が i-1 のアクセスが必要であることから、ループ伝搬の流れ依存があり単純に別の名称の配列を用意するだけでは依存関係を除去できない。ただし、dd を計算するための…=u(i,j,k)参照のあとの代入 u(i,j,k)=…が出力依存である。u(i,j,k)の参照を終了するまでデータの書き込みができず、速度低下の要因となる。

この出力依存を解決するためには、別の配列 u2 を用意する。別配列 u2 をもとの配列 u に書き戻す手間を考えると、i ループを 2 段アンローリングすると効率がよい。図 8 に、それを示す（ただし簡単化のため、単純 k,j,i ループコードに適用した場合を示す）。

```
      do k = 1, nz
        do j = 1, ny
          do i = 1, nx, 2
            dd = (u(i,j,k)-u(i-1,j,k))*dxinv
     &         + (v(i,j,k)-v(i,j-1,k))*dyinv
```

```
&               + (w(i,j,k)-w(i,j,k-1))*dzinv
            dp = beta*dd
            u2(i,j,k)   = u(i,j,k)   + dtdx*dp
            u2(i-1,j,k) = u(i-1,j,k) - dtdx*dp
            v(i,j,k)   = v(i,j,k)   + dtdy*dp
            v(i,j-1,k) = v(i,j-1,k) - dtdy*dp
            w(i,j,k)   = w(i,j,k)   + dtdz*dp
            w(i,j,k-1) = w(i,j,k-1) - dtdz*dp
            p(i,j,k)   = p(i,j,k) + dp
            err = max (err, abs(dd))

            dd = (u2(i+1,j,k)-u2(i,j,k))*dxinv
&               + (v(i+1,j,k)-v(i+1,j-1,k))*dyinv
&               + (w(i+1,j,k)-w(i+1,j,k-1))*dzinv
            dp = beta*dd
            u(i+1,j,k)   = u2(i+1,j,k)    + dtdx*dp
            u(i,j,k)     = u2(i,j,k)      - dtdx*dp
            v(i+1,j,k)   = v(i+1,j,k)     + dtdy*dp
            v(i+1,j-1,k) = v(i+1,j-1,k)   - dtdy*dp
            w(i+1,j,k)   = w(i+1,j,k)     + dtdz*dp
            w(i+1,j,k-1) = w(i+1,j,k-1)   - dtdz*dp
            p(i+1,j,k)   = p(i+1,j,k)     + dp
            err = max (err, abs(dd))

c           --- u2の結果を保存
            u(i-1,j,k) = u2(i-1,j,k)
         enddo
       enddo
```

図 8　出力依存無しｉループ２段アンローリングコード

図 8 のコードでは、配列ｕに関連するｉループについて展開したが、同様のアイデアを配列ｖのｊループ、配列ｗのｋループについても同時に適用することができる。すなわち、余分な配列を３つ用意できれば、ループ中から出力依存が完全に除去できる。図 8 のコードは、データのプリフェッチなど、コンパイラによる最適化がうまく働く場合は高速化される可能性がある。

３．性能評価

（１）評価環境

T2K オープンスパコン（東大）(HITACHI HA8000 クラスタシステム）のノードは、AMD Opteron 8356 (2.3GHz, 4 コア)を 4 台（4 ソケット）搭載しており、メモリは 32GB である。理論最大演算性能は、ノードあたり 147.2GFLOPS である。キャッシュサイズは、L1 命令キャッシュ、L1

データキャッシュともに64Kbytesであり、2 Way Associativity（ライトバック、3サイクル）である。また、L2キャッシュは512Kbytesである。L1キャッシュとL2キャッシュはコアごとに独立している。L3キャッシュ 2048Kbytesであり、ソケット内で共有されている。各ソケットには、ローカルメモリ8GBがある。ソケット間はHyperTransportという高速通信網で連結されており、ローカルメモリへのデータアクセス時間はソケットからの距離により異なる。ただし、ソケット間のキャッシュデータの一貫性はハードウエア的に保障される。ccNUMA（cache coherent non-uniform memory access）型のアーキテクチャ構成である。

通信性能は運用クラスタ群で異なる。ここでは Miri-10G が 4 本実装されており、最大で5GB/secの双方向性能を有するA群を利用している。

コンパイラは、日立最適化Fortran90 V01-00-/Bで、コンパイラオプションは、コンパイラオプションとして -precexp=4 -autoinline -opt=ss -noparallel を指定した。実験期間は、2009年2 月6日～2月13日である。

（２）ベンチマークプログラム

MPIによる並列化がなされた3次元津波伝搬コード[4]において、MPI通信部分を除去し1コア版にしたプログラムを利用する。問題サイズは、nx=256, ny=256, nx=50 である。ここでは海底の深さは均一とし、k=10～48である。シミュレーション開始から直後の、主要カーネルを含むホットスポットにおいて9回反復の時間を計測した。

（３）結果

ブロック化コードのブロック幅kblkは32, 64, 128, 256と変化させた結果、256が最速であったため、すべてこの値を利用した。すなわち問題空間全体を1つのブロックとするが、ベンチマークプログラムでは海底の深さが均質であるため、kblk = nx = 256 が高速となるのは当然の結果といえる。

図 9に各コードの実行時間を載せる。図 9から以下のことがわかる。

- ＩＦ文除去コードは、論理マスクコードより遅い。理由は、ＩＦ文除去コードは最内ループがkループとなり、これは配列の第3要素であることから、不連続アクセスとなりデータアクセス時間が増大することによる。
- 単純k,i,jループは、論理マスクコードに比べて演算量が増えているのにもかかわらず、論理マスクコードより高速である。このことは、ループ中のIF文の実行オーバヘッドが大きいことを意味している。
- 単純ブロック化コードは、単純k,j,iループコードより高速である。これは、余分な演算を行わないことによる。
- w配列局所化は、1%程度の速度向上に寄与する。
- ＩＦ文除去コードに、配列のならびをu(i,j,k) からu(k,i,j)に変更し連続アクセス化したコードは、論理マスクコードより遅い。この理由は、kループが10～48に変化するが、実配列は1～50まで確保されており連続アクセスの効率が悪いことによる。

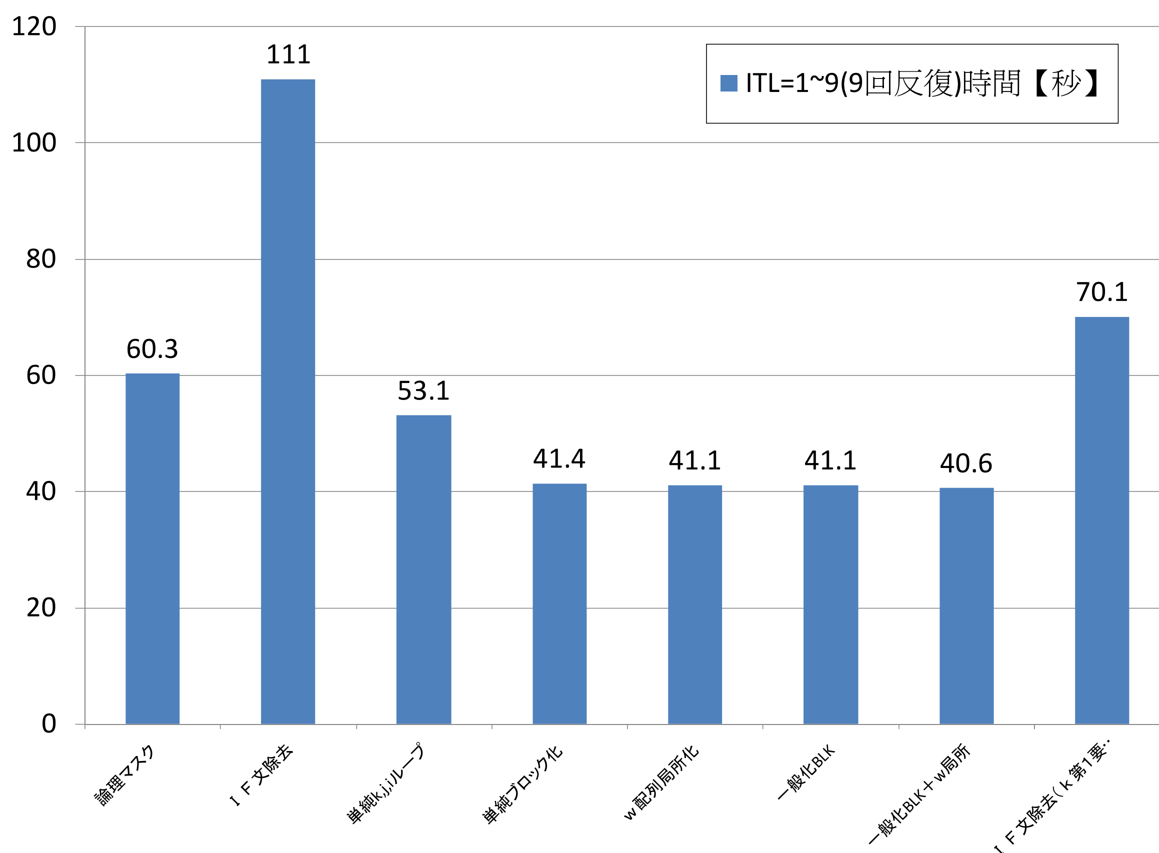

図 9　各コードの実行時間

以上の結果より、このベンチマークでは論理マスクコード（60.3 秒）に対して、一般化ブロック化+W 局所化コード（40.6 秒）が 48%の速度向上、ＩＦ文除去コード（111 秒）に対しては、同コードが 270%の速度向上を達成した。なお、一般化ブロック化+W 局所化コードの 1 コアでの論理ピーク性能（9.2GFLOPS）に対するカーネル部分のみの演算効率は、14.5%（1.34GFLOPS）であった。

3．おわりに

本稿では、3 次元津波伝搬シミュレーションコードにおける最適化手法を紹介し、ベンチマークコードに対し性能を測定した。kblk*kblk の正方領域内の海底情報を利用して k ループを連続化する方法（ブロック化コード）を紹介し、少なく見積もって T2K オープンスパコンで 50%程度の速度向上の可能性があることをベンチマークプログラムから示した。

本稿で紹介したブロック化コードは、陸地・浅瀬があるような地形区間での計算時には高速化が実現できない。しかし MPI 並列化されたコードにおいては、担当区間に海だけの領域が存在する。この場合、陸地担当のプロセスに対して、海だけ担当のプロセスは演算量が多い（＝すなわち、k ループ長が長い）はずであり、かつ、全体の実行時間が海だけ担当のプロセスの実行時間に拘束されるはずである。したがって MPI 並列化では、ブロック化コードによる速度向上の恩恵が大きいと予想する。

今後、実海底データによる津波伝搬シミュレーションコードにブロック化コードを適用し性

能評価を行う予定である。

謝辞

3 次元津波伝搬シミュレーションのソースコード一式をご提供いただいた、東京大学地震研究所　齊藤竜彦博士、古村孝志教授に深く感謝いたします。なお本研究は、東京大学情報基盤センター平成２０年度Ｔ２Ｋオープンスパコン（東大）共同利用研究プロジェクトの一環で行われた。

参　考　文　献

[1]古村孝志, 地球シミュレータによる地震の強い揺れと津波の予測・災害軽減,計算工学, 13, 2, 14-17, 2008.

[2]Furumura, T. and T. Saito, An integrated simulation of ground motion and tsunami for the 1944 Tonankai earthquake using high-performance super computers, Journal of Disaster Research, Vol 7, No2, in press, 2009.

[3]古村孝志・今井健太郎・齊藤竜彦 , 南海トラフ連動型巨大地震による地震動と津波の予測, 月刊地球, 印刷中, 2009.

[4]Saito, T. and T. Furumura, Three-dimensional simulation of tsunami generation and propagation: application to intraplate events, J. Geophys. Res.,114, B023207, doi:10.1029/2007JB005523, 2009.

[5] C.W.Hirt, B.D.Nichls, and N.C.Romero, SOLA-A Numerical Solution Algorithm for Transient Fluid Flows, UC-34 and UC-79d, Los Alamos Scientific Laboratory, 1975.