解　説

ミニスーパコンピュータ Convex SPP2000 を用いた利用法および処理例

吉岡　良雄
理学部情報科学科
(slyoshi@si.hirosaki-u.ac.jp)

成田　好輝
理学部情報科学科大学院生
(総合情報処理センター技術補佐員)
(ynarita@cc.hirosaki-u.ac.jp)

1　まえがき

平成６年度に情報処理センターから省令施設の総合情報処理センターへの昇格とともに、計算機システムの更新作業が始まった。この更新作業において、平成５年度第１次補正予算によるキャンパス情報ネットワークが整備されたこともあり、大学内の各研究室からこのネットワークを経由して容易に利用できる計算サーバを導入することを前提に仕様書を作成した。すなわち、グラフィックユーザインタフェースが利用できること、使い易いOSであるということ、２４時間稼働を目指しているので省エネルギー化になっていること、などである。このような仕様書によって、日本電気からミニスーパコンピュータ Convex SPP2000 (8CPU) が提案され、入札の結果日本電気が落札し、この Convex SPP2000 が導入される運びとなった。その主な仕様は以下のようになっている。

プロセッサ：	Convex SPP Exemplar (8CPU)
ネットワークインタフェース：	ギガスイッチに 100Mbps SAS で接続
１ＣＰＵ当たりの性能：	124MIPS、198MFLOPS
トータル性能：	992MIPS、1.584GFLOPS
メモリバンド幅：	1.25GB/秒
メモリ：	512MB
ディスク容量：	20GB
オペレーティングシステム：	UNIX(HP/UX)
言語：	自動並列化対応 FORTRAN、 並列化対応 C 言語
開発支援ツール：	並列化対応プロファイラ CXpa
ライブラリ：	並列化対応数値計算ライブラリ
バッチシステム：	NQS

グラフィックユーザインタフェース：X Window System X11R5

この仕様からも理解できるように、8CPU からなる並列処理コンピュータでありスーパコンピュータ並みの性能があることが分かる。大規模な計算を行うユーザにとっては、非常に頼もしい計算機であると考えている。この 8CPU のうち、4CPU はバッチ処理用、後の 4CPU は会話型処理用に分けて利用できるようになっている。

上述のように、平成７年２月に２４時間稼働のミニスーパコンピュータ Convex SPP2000 を導入し１年が経過した。この間、Convex の利用法がよく分からなかったこともあり、利用されることが少なかった。平成７年度後半になって情報科学科の修士論文のまとめの時期に入り、情報認識のシミュレーションや画像処理などを進めるに当たり、情報科学科のワークステーションで計算するよりは Convex を利用すれば数十倍の速さで処理できるという噂が広がり Convex の利用が急に増えはじめた。このようなことから、ワークステーションで大規模計算を行うよりは、Convex を利用して大規模計算を行った方が速いということが分かる。

本報告では、導入されたミニスーパコンピュータ Convex SPP2000 の処理例を上げながら、その利用法を述べることにする。本報告によって、Convex のユーザが増えれば幸いに思う。

2　処理例

ここでは、FORTRAN および C 言語による処理例を示しながら利用法を述べる。まず、研究室の端末は X 端末または PC において X 端末エミュレータを利用することが望ましい。この端末から、"rlogin hakkoda" によって Convex にログインして、次のコマンドを入力する。

```
setenv DISPLAY my_display_IP:0.0
```

ここで、my_display_IP は、自分の研究室にある X 端末の IP アドレスである。以下において、FORTRAN および C 言語での処理例を示す。

(1) FORTRAN での処理例

プログラムをコンパイルするには、次のコマンドを入力する。

```
fc -cxpa -O3 file.f      (file.f はコンパイルしたいファイル名を入力)
```

ここで、開発支援ツール CXpa を利用するにあたり、次の３つのオプションを選択できる。

-cxpa:　ルーティン、ループおよび並列領域での分析
-cxpar:　ルーティンのみでの分析
-cxpab:　ベーシックブロックのみの分析

また、-On(n=0,1,2,3)オプションを用いることにより、コンパイル時の最適化の度合いを定義することができる。各数字における、最適化の度合いは次の通りである。

-O0: 実行時間長およびスカラー最適化
-O1: -O0 に加えて、プログラムレベル最適化およびレジスタ配置最適化
-O2: -O1 に加えて、命令スケジューリング、ソフトウェアパイプラインおよびデータ配置最適化
-O3: -O2 に加えて、並列最適化

最適化オプションが省略されると、-O2 オプションが自動的に選択される。

コンパイル終了後、以下のコマンドを入力し、CXpa ツールを起動する。

```
cxpa a.out &
```

このコマンドを入力すると、図1に示す CXpa Executable Manager Window 画面が出力される。

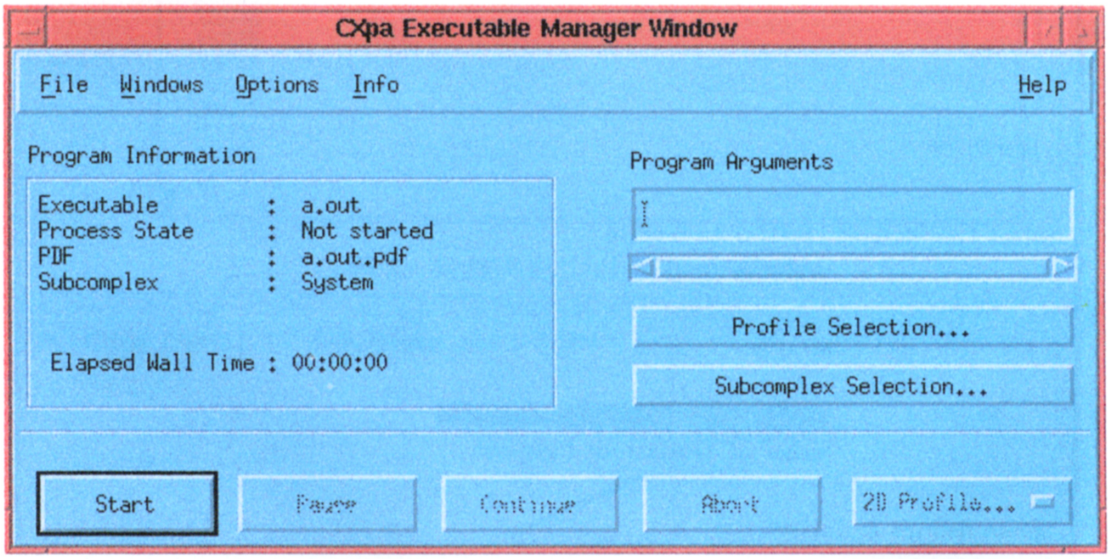


図 1 CXpa Executable Manager Window の画面

入力データなど、プログラムに渡す必要があるパラメータが存在するときは、Program Arguments にその内容を記述する。標準入力からの入力データがある場合には、例えば「< file.dat」のように入力する。

「Profile Selection...」と書かれたバーをクリックすると、Profile Selection Dialog が表示される。Profile Selection Dialog では、画面上部に表示されているトグルボタンをクリックすることで、関数ごとにどのような情報を収集するかを決定することができる。

また、Processor Event(s)の各項目の中から１つ選択することで、ローカルキャッシュミスやリモートキャッシュミスなどの情報を表示できるようになる。

図２に、Profile Selection Dialog の画面を示す。

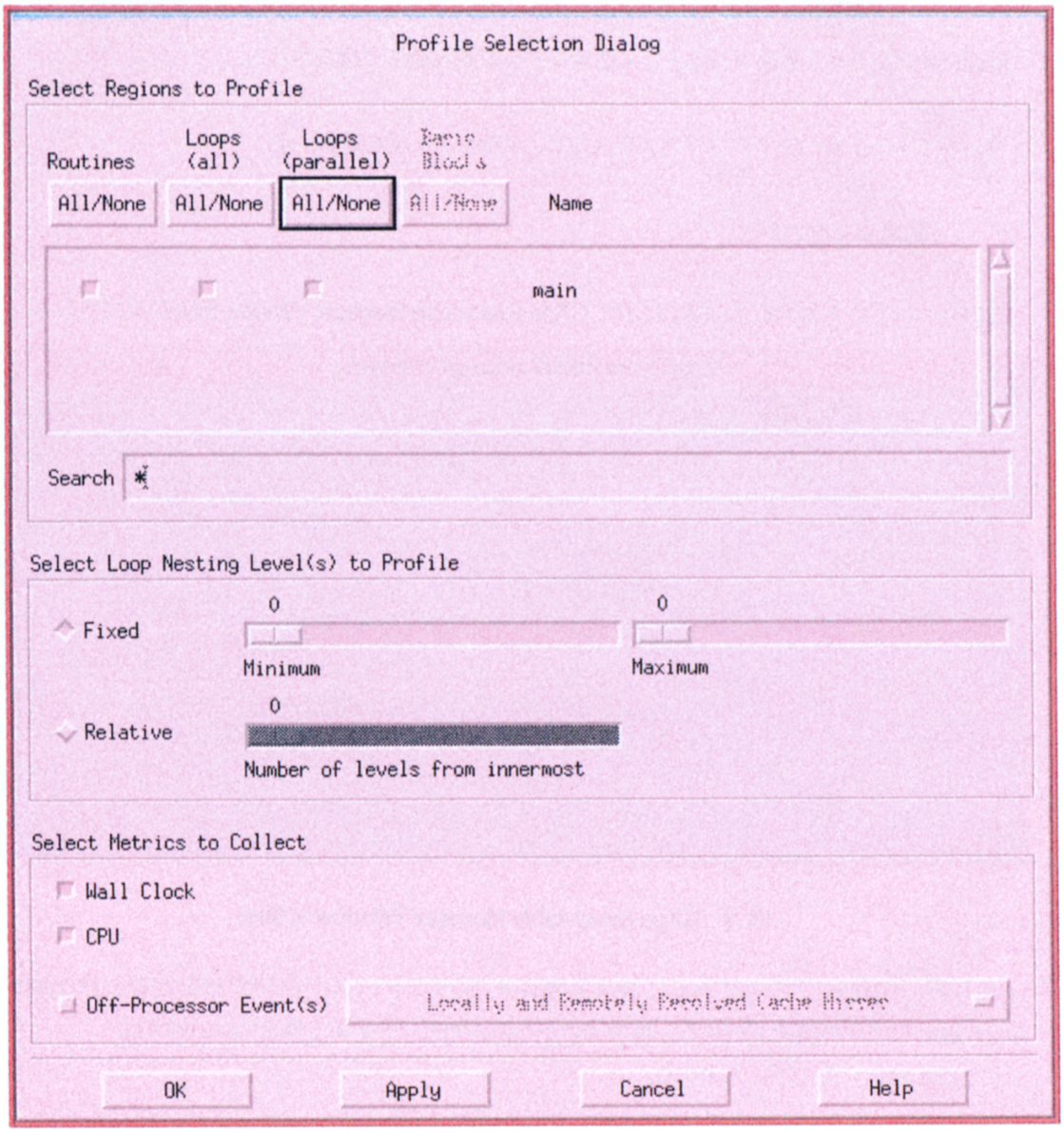


図 2　Progile Selection Dialog の画面

変更した内容を適用するためには、Apply ボタンを押し、Proifle Selection Dialog を終了するには、OK ボタンを押す。

以上で、収集したい情報の定義が終了したので、CXpa Executable Manager Window に戻る。

ここで、start ボタンを押すと、プログラムの実行結果を出力する xterm ウィンドウが表示される。

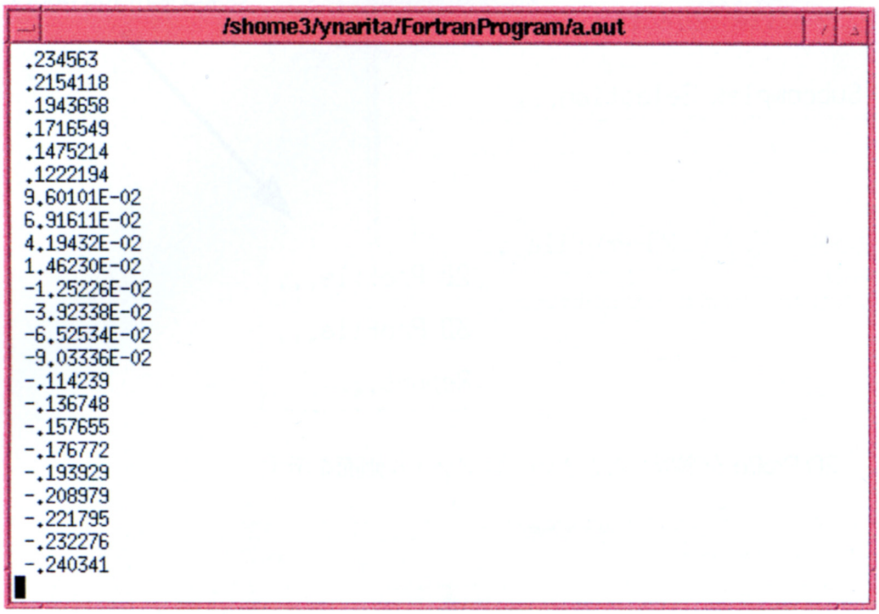
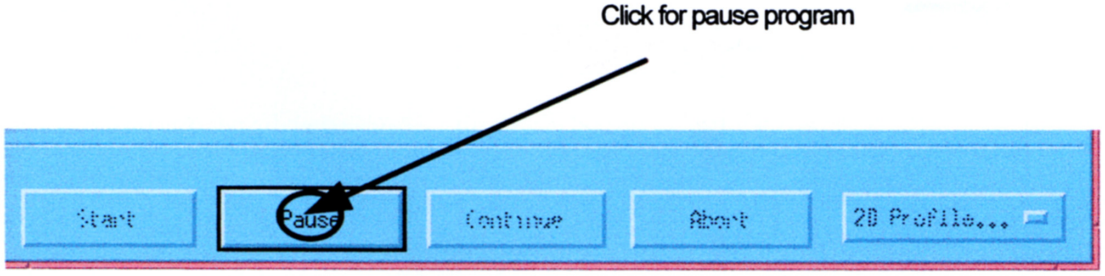


図 3　実行結果を出力する Xterm Window

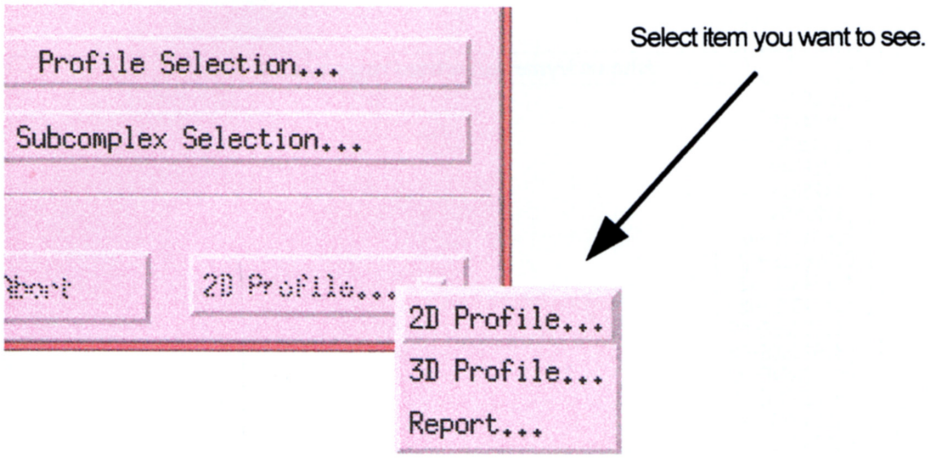
また、CXpa が動作している間、Pause ボタンを押すことで、プログラムの実行を一時中断することができる。



CXpa の解析が終了すると、以下の作業が実行できるようになる。

- ２次元、３次元による Profile Graph の表示
- レポート形式によるプログラム実行の解析結果の表示

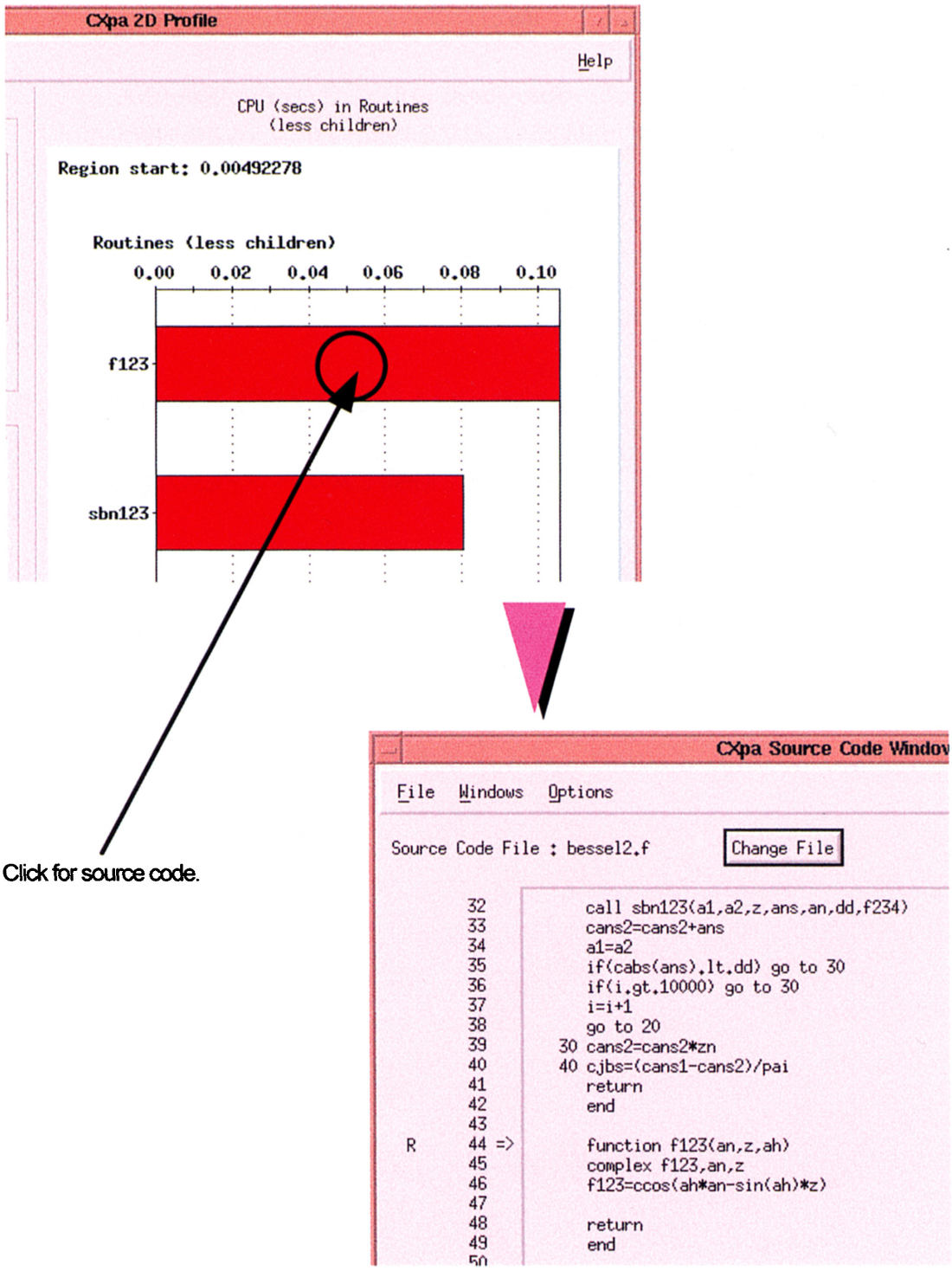
CXpa Executable Manager Window の右下にあるボタンや、画面上部にある、Window メニューの中から、2D Profile などを選択することで、おのおのの情報を表示することができる。



以下に、2D Profile を選択したときのプロファイル画面を示す。

図 4　2D Profile Window の画面

この画面は、後で示すFORTRAN Programの各関数における、CPU timeを示している。この画面上に表示されている棒グラフを、マウスの左ボタンでクリックすることにより、その関数のソースプログラムを表示することができる。



(2) C 言語での処理例

次に、C 言語での処理例を示す。プログラムをコンパイルするには、次のコマンドを入力する。

```
cc -cxpa -O3 file.c     (file.c はコンパイルしたいファイル名を入力)
```

FORTAN の処理例同様、-cxpa、-cxpar、-cxpab の３つのオプションが選択できる。
また、最適化オプション -On (n=0,1,2,3)は、C 言語の場合、次のような意味を持つ。

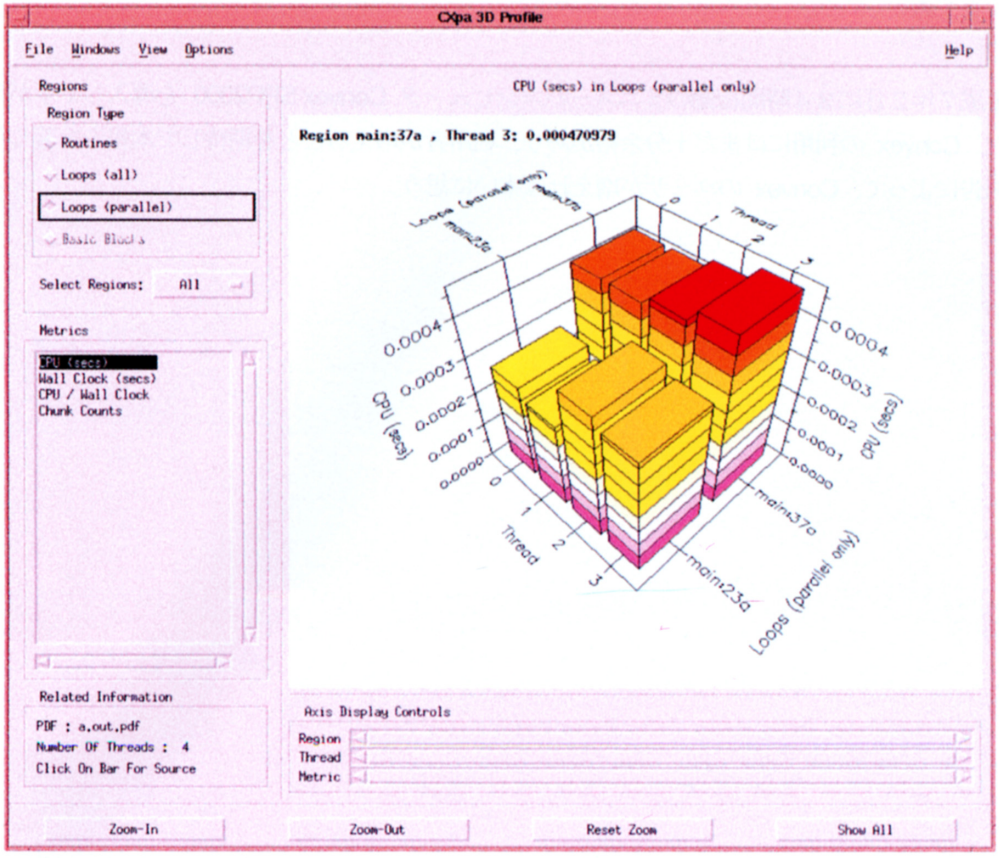
-O0 : 実行時間長、スカラー最適化
-O1 : O0 に加えて、プログラムレベル最適化および命令スケジューリング
-O2 : O1 に加えて、データ配置最適化およびベクトル化
-O3 : O2 に加えて、並列最適化

FORTRAN 同様、最適化オプションが省略されると、 -O2 オプションが自動的に選択される。

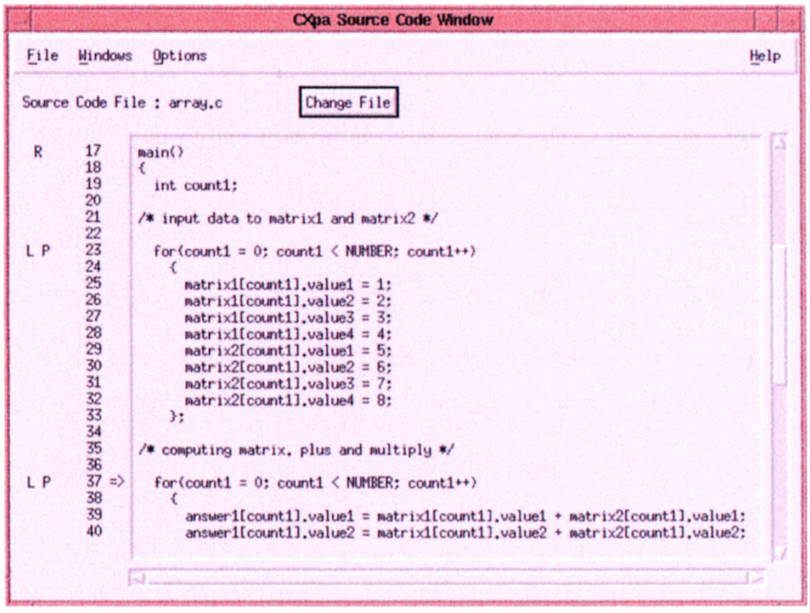
Cxpa ツールを起動するには、FORTRAN の時と同様に、次のコマンドを入力する。

```
cxpa a.out &
```

ツール起動後の操作は、FORTRAN の処理例に示したものと同様である。ここでは、後に示す C program の解析結果を 3D-Profile で表示したものを示す。



この図では、main 関数の２３行目と３７行目が並列化されていて、４つのスレッド（CPU）に計算が分配されていることが示されている。グラフの縦軸は、おのおのの処理に CPU が消費した時間を示している。



3　まとめ

平成７年２月に２４時間稼働のミニスーパコンピュータ Convex SPP2000 を導入し１年が経過した。Convex の利用にはまだ十分余裕があり、利用者が増えることを期待して本報告を著した。本報告によって、Convex のユーザが増えれば幸いに思う。

付録１：本報告で使用したFORTRAN　Program

```
c       Bessel function Jn(z)
        function cjbs(an,z,dd)
        complex cjbs,an,z,cans1,cans2,zn,ans,f123,f234
        external f123,f234
        pai=3.14159265
        cans1=cmplx(0.0,0.0)
        cans2=cmplx(0.0,0.0)
        a1=0.0
        do 10 i=1,10
        a2=0.1*float(i)*pai
        call sbn123(a1,a2,z,ans,an,dd,f123)
        cans1=cans1+ans
        a1=a2
   10 continue
        zn=csin(pai*an)
        if(cabs(zn).lt.dd) go to 40
        a1=0.0
   20 a2=exp(0.1*float(i))
        call sbn123(a1,a2,z,ans,an,dd,f234)
        cans2=cans2+ans
        a1=a2
        if(cabs(ans).lt.dd) go to 30
        if(i.gt.10000) go to 30
        i=i+1
        go to 20
   30 cans2=cans2*zn
   40 cjbs=(cans1-cans2)/pai
        return
        end

        function f123(an,z,ah)
        complex f123,an,z
        f123=ccos(ah*an-sin(ah)*z)
        return
        end
```

```
      function f234(an,z,ah)
      complex f234,an,z
      fff=0.5*(exp(ah)-exp(-ah))
      f234=cexp(-ah*an-fff*z)
      return
      end

c     integration
      subroutine sbn123(a1,a2,z,ans,an,dd,func)
      complex x1,x2,x3,x4,an,ans,z,func
      x1=CMPLX(0.0,0.0)
      b1=0.5*(a1+a2)
      x2=func(an,z,b1)
      x3=func(an,z,a1)+func(an,z,a2)
      m=2
      h=0.25*(a2-a1)
   10 x4=cmplx(0.0,0.0)
      do 20 i=1,m
      b1=h*(2.0*float(i)-1.0)+a1
      x4=x4+func(an,z,b1)
   20 continue
      ans=h*(x3+2.0*x2+4.0*x4)/3.0
      d1=cabs(ans-x1)
      if(dd.gt.d1) go to 30
      x1=ans
      x2=x2+x4
      h=0.5*h
      m=2*m
      go to 10
   30 return
      end
```

付録２：本報告で使用したC言語Program

```
/* Matrix computing Program */

#include<stdio.h>
#include<math.h>
#define NUMBER 1000

struct Matrix{
   int value1;
   int value2;
   int value3;
   int value4;
};

struct Matrix matrix1[NUMBER], matrix2[NUMBER];
struct Matrix answer1[NUMBER], answer2[NUMBER];

main()
{
   int count1;

/* input data to matrix1 and matrix2 */

   for(count1 = 0; count1 < NUMBER; count1++)
      {
         matrix1[count1].value1 = 1;
         matrix1[count1].value2 = 2;
         matrix1[count1].value3 = 3;
         matrix1[count1].value4 = 4;
         matrix2[count1].value1 = 5;
         matrix2[count1].value2 = 6;
         matrix2[count1].value3 = 7;
         matrix2[count1].value4 = 8;
      };

/* computing matrix. Addition and multiplication */
```

```
  for(count1 = 0; count1 < NUMBER; count1++)
    {
      answer1[count1].value1 = matrix1[count1].value1 + matrix2[count1].value1;
      answer1[count1].value2 = matrix1[count1].value2 + matrix2[count1].value2;
      answer1[count1].value3 = matrix1[count1].value3 + matrix2[count1].value3;
      answer1[count1].value4 = matrix1[count1].value4 + matrix2[count1].value4;

    answer2[count1].value1 = matrix1[count1].value1 * matrix2[count1].value1 +
matrix1[count1].value2 * matrix2[count1].value3;
    answer2[count1].value2 = matrix1[count1].value1 * matrix2[count1].value2 +
matrix1[count1].value2 * matrix2[count1].value4;
   answer2[count1].value3 = matrix1[count1].value3 * matrix2[count1].value1 +
matrix1[count1].value4 * matrix2[count1].value3;
    answer2[count1].value4 = matrix1[count1].value3 * matrix2[count1].value2 +
matrix1[count1].value4 * matrix2[count1].value4;
};

/* Display Answer */
  for(count1 = 0; count1 < NUMBER; count1++)
    {
      printf("answer1[%d] = [%d %d %d %d]₩n", count1,
              answer1[count1].value1,
              answer1[count1].value2,
              answer1[count1].value3,
              answer1[count1].value4);

      printf("answer2[%d] = [%d %d %d %d]₩n", count1,
              answer2[count1].value1,
              answer2[count1].value2,
              answer2[count1].value3,
              answer2[count1].value4);
    }
}
□
```