数 値 解 析 28－4
（１９８９．３．１４）

クレイ・スーパコンピュータにおける マルチタスキング機能

斎藤 務

日本クレイ（株） マーケティング部

本報告では、クレイ・リサーチ社のマルチタスキング手法について、昨年（１９８８）発表されたオートタスキングを中心に解説する。オートタスキングに於いては、FORTORAN ソースコードから単一のコマンドによって複数のプロセッサーで並列処理可能な実行モジュールを生成することが出来、従来の手法に比べて格段に使いやすいものとなっている。
オートタスキングの中核をなすプログラム解析ルーチンは、並列処理コードの生成のみならず、ベクトル化促進機能、あるいはサブプログラムのコーリングルーチンへの展開（インライン展開）といった様々な最適化機能を備えておりそれらについても紹介する。

MULTITASKING ON CRAY SUPERCOMPUTERS

Tsutomu Saito
Marketing Dept.,
CRAY RESEARCH JAPAN LTD., Ichbancho Eight-One Bldg.,
4th Floor, 6-4 Ichiban-cho, Chiyoda-ku, Tokyo 102, Japan

Cray Research Inc., has introduced a new method of parallel processing called "Autotasking" last year 1988. This report describes the new software together with ordinary multitasking methods of Macrotasking and Microtasking. Autotasking compilar system automatically generates executable codes for parallel processing by invoking just a single command line.
The dependency analyzer of autotasking compiler system also has capabilities of enhanced vectorization and in-line expantion of sub-programs into their calling routines. These functions altogether work for producing highly optimized code. This report also demonstrates some of those new functions.

1. はじめに

クレイ・リサーチ社では、１９８２年に、主記憶共有型多重プロセッサスーパーコンピュータシステム X-MP の発売以来、様々な並列化処理機能の開発をつづけている。複数のジョブを複数のプロセッサで処理しシステムスループットを向上させるマルチプロセッシングや、一つのジョブを複数のプロセッサで処理して各ジョブのスループットを向上させるマルチタスキングがそれらに属する。マルチタスキングは、ベクトル、スカラー処理の別なく使うことが出来る事、アムダールの法則[1]の制限はあるものの、並列度の高いプログラムに対しては、使用するプロセッサの数にほぼ比例したスループットの向上が期待できる事などからスーパコンピューティングにおいて最近注目を集めており今後も引き続いて研究開発されて行くものと思われる。クレイ・リサーチ社では既にマクロタスキング、マイクロタスキングというマルチタスキング機能を世におくりだしているが、これらを第一、第二世代のマルチタスキング手法とするならば第三世代ともいうべきオートタスキングが昨年の１１月に発表された。本報告では、マクロ、マイクロタスキングについて簡単に紹介した上で、このオートタスキングについて例を交えながら解説を行う。

2. マクロタスキング

現在クレイでマクロタスキングと呼ばれている第一世代の並列処理手法ではFORTRANプログラムをサブルーチンレベルで分割し、それらの実行を複数のCPUで同時に行う。ユーザは並列処理するサブルーチンを実際にソースプログラムへとコーディングする等の作業を行う必要がある。また、これら並列処理プロセス間で実行時に同期をとる必要のあるときには、それらの機能を持つライブラリーをコールする必要があり、ユーザにかなりの負担を強いることになる。このような理由から、比較的大きなオーバーヘッドとも相まってあまり一般的な方法とはなっていない。しかしながら、並列実行可能な部分の計算量が大きい（グラニュラリティが大きい）プログラムに対しては、非常に有効で、現在でも気象関連のアプリケーション等に使われて成果を上げている。

マクロタスキングのライブラリーには、その機能別に、１）複数のタスクの発生・消滅を制御するためのタスク制御ルーティン、２）タスク間の同期をとるイベント制御ルーティン、３）コモンデータ領域を、他のタスクから保護するロック制御ルーティンの三種類があり図１にそれらの概念図を示す。図１－ａにある様に、ライブラリー TSKSTARTによって、子プロセス:TASK1 が生成され、また、TSKWAITによって親プロセス:TASK0は、TASK1の終了を待ってそれを消滅させる。一方、図１－ｂでは、ライブラリー EVPOST, EVWAIT, EVCLEARなどが、並列に実行しつつあるプロセス間で同期をとるために使われている様子が示されている。また図３－ｃでは、論理的に並列処理してはいけないプログラム部分をロック制御ルーティンを使って保護している様子を示している。TASK0の CALL LOCKON と CALL LOCKOFF ではさまれた部分が実行中は、TASK1 は実行を休止し、TASK0 の LOCKOFF を待って実行を開始する。図３－ｃでは、たまたま TASK0で TASK1より早くロックされた場合を示しているが、もし TASK1 のロックが先にかかった場合にはTASK0 がTASK1で LOCKOFF されるまで待状態になる。

マクロタスキングの実例を図２に示す。この例では、２つのループを含むプログラムが２つのCPUで並列処理されている。親プロセス側の TASKSTART、TASKWAIT によって、サブルーチンTが、子プロセスとして生成、実行、消滅させられており、２つのプロセス間で、DO 10 ループが完了してから DO 30 ループへ実行が移るように、EVPOST, EVWAIT、EVCLEAR の各ライブラリーコールによる同期がはから

れている。

3. マイクロタスキング

マクロタスキングがサブルーチンレベルの並列処理を行ったのに対し、マイクロタスキングは FORTRANプログラムの DOループレベルで並列処理を行う。ユーザは、コンパイラ指

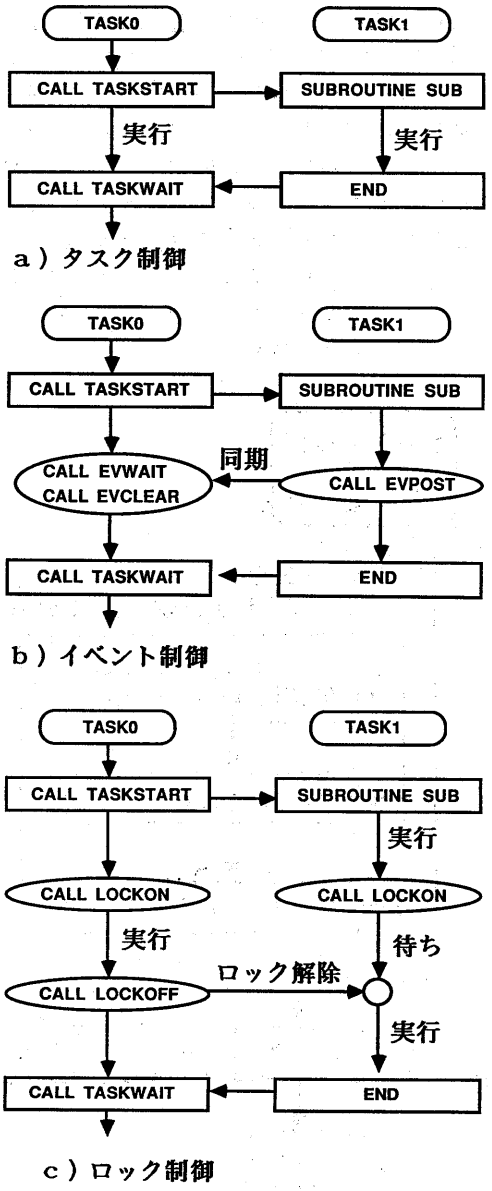


図 1. マクロタスキングライブラリ

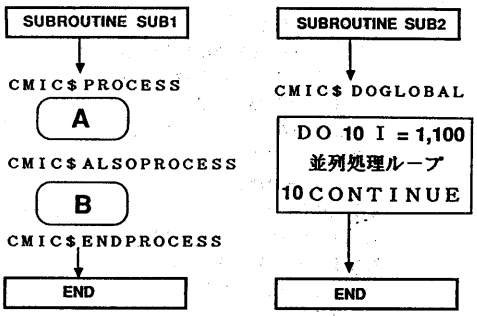
```
      PROGRAM MAIN
      COMMON/EVENTS/IDONE1,IDONE2
      CALL EVASGN ( IDONE1 )
      CALL EVASGN ( IDONE2 )
      CALL TSKSTART ( IDTASK, T)
      DO 10 I = 1,N/2
          A(I) =  AINIT(I)
   10 CONTINUE
      CALL EVPOST ( IDONE1 )
      CALL EVWAIT ( IDONE2 )
      CALL EVCLEAR ( IDONE2 )
      DO 30 J =  1,N/2
          DO 20 I = 1,N
              B(I,J) =  A(I)*C(J)
   20     CONTINUE
   30 CONTINUE
      CALL TSKWAIT ( IDTASK )
          .
      STOP
      END

      SUBROUTINE T
      COMMON/EVENTS/IDONE1,IDONE2
      DO 10 I = N/2+1,N
          A(I) = AINIT(I)
   10 CONTINUE
      CALL EVWAIT ( IDONE1 )
      CALL EVCLEAR ( IDONE1 )
      CALL EVPOST ( IDONE2 )
      DO 30 J = N/2+1,N
          DO 20 I = 1,N
              B(I,J) = A(I)*C(J)
   20     CONTINUE
   30 CONTINUE
      RETURN
      END
```

図 2. マクロタスキングプログラム例

示文と同様なマイクロタスキング指示行をソースプログラム中に挿入する事によって、並列処理可能な部分の指定を行う。

図3にマイクロタスキングの代表的なふたつの形態を示す。図3－aの様にサブルーチン内に独立して実行し得る部分が続いてある場合、CMIC$ で始まるマイクロタスキング指示文を置くことによってこれらを複数のCPUで並列処理することができる。また図3－bに示すように DOループ の繰り返し計算が、繰り返しごとの依存性を持たないときには、やはり指示文を一行加えることによってDOループの複数CPUによる処理が行われる。



a）A,Bが独立の場合　　b）並列処理可能な DO ループ

図 3. マイクロタスキング

この手法は、マクロタスキングに比べて並列処理の準備のためのオーバーヘッドが非常に低く、ユーザの負担も少ないため第二世代の並列化手法として幅広く使われている。

4.オートタスキング

オートタスキングは、マクロ、マイクロに続くクレイの第三世代並列処理手法で、これによってユーザの介在無しに、並列処理を行うことが可能となった。オートタスキングは、その並列処理手法自体はマイクロタスキングと同じであるが、cf77という単一コマンドによってFORTRANプログラムの並列処理可能な部分の検出および指示文の挿入から、最終的には、複数のCPUによって実行可能なコードを生成するところまでを自動的に行うことができる。更に必要であれば、オートタスキングの過程で出力される様々なメッセージを参考に、ユーザ自身が指示文を追加してより効率のよいコードを作成することも可能である。オートタスキングは、クレイのFORTRANコンパイラー cft77 Release 3.0 と共に提供されるクレイの標準ソフトウェアで、ループの書換えや分割を含むベクトル化に対する最適化機能、あるいはマトリックス演算を認識してこれを高速なライブラリーコールに置き換えたり、サブルーチンのコーリングルーチンへの展開と言った最適化機能も強化されている。以下にこのcft77 Release 3.0 を核としたオートタスキングコンパイラシステムについて少し詳しく解説する。

4.1オートタスキングコンパイラシステム

オートタスキングFORTRANコンパイラシステムは、図4に示すように依存性解析ルーチン(fpp)、コード変換ルーチン(fmp),そしてコード生成ルーチン(cft77)という三つの部分からなり、cf77 というコマンドを用いて起動することができる。 cf77 はオプションの指定によってこれら三つのプログラムをすべて実行してマルチタスク用のコードを直接生成することも、また必要に応じてそれらを別々に実行することもできる。cf77 のオプションを図5に示すが、例えば、

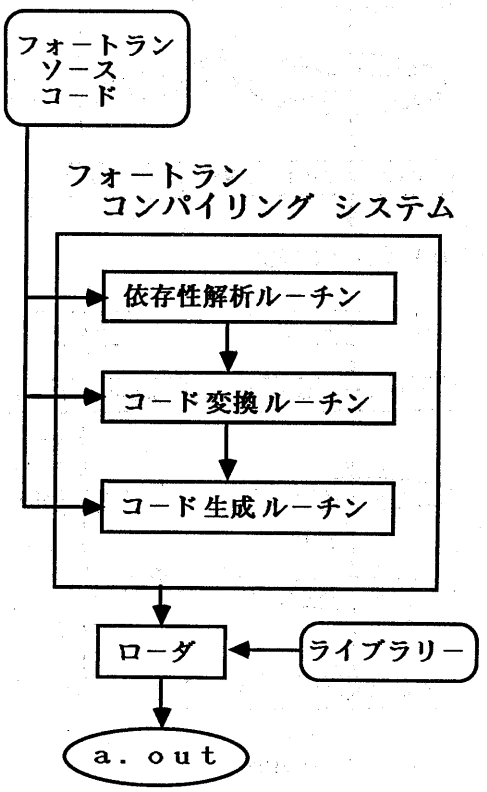


図 4. オートタスキング
コンパイリングシステム

```
% cf77 -Zp souce.f
```

とすれば、複数のCPUで実行可能なコードがa.outとして自動的に作られるので、ユーザは適当な入力データと共にただちにこれを実行することができる。また、

```
% cf77 -Zv source.f
```

とすれば、並列化は行われず、ベクトル化に対する最適化のみが施されたコードが生成される。そのほか、オプションの指定によって、三つのプログラムの様々な組合せができることが図5に示されている。

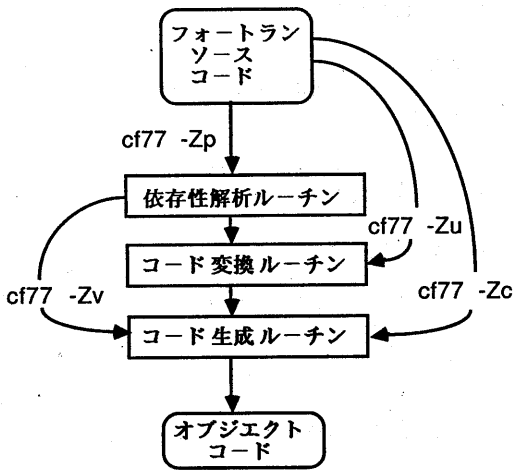


図 5. cf77 オプション

4.1.1 依存性解析ルーチン(fpp)

依存性解析ルーチン fpp は入力されたFORTRANソースコードを解析してループの繰り返し数、依存性の有無、ループ内の計算量等を基準に、並列処理可能な部分を検出し、必要な指示文を挿入して行く。オートタスキングの並列処理原理自体はマイクロタスキングと同様、サブルーチン内の並列処理可能部分、例えば依存性のないDOループ等(図3参照)に対して行われるもので、その指示文はマイクロタスキングのそれと同様の概念を持っている。実際オートタスキングとマイクロタスキングでは、共通の指示文もいくつかあるが前者は種類も多く、よりきめ細かな並列処理制御が可能となっている。

オートタスキングはまたマクロあるいはマイクロタスク化されたプログラムに対しても有効である。fppにこれらの処理がなされたプログラムが入力されると、既に指示文の入ったサブルーチンに対しては並列化解析を行わないが、その他のサブルーチンに対してはより一層の並列化を目的としたオートタスキングが行われる。

fppは入力されたコードをサブルーチン単位で解析し、幾つかのサブルーチンにわたっての並列化解析は行わない。従って並列化の可能性について情報が不足することがある。例えば、図6のプログラム例に示すように、DOループ内にサブルーチンコールがあるような場合、このコールされたサブルーチンがコーリングサブルーチンの繰り返しに対して依存性を持つかどうかの判定は、今の所fppには下すことができない。この様なとき安全のため、fppはこのループの並列化処理はあきらめてその理由を述べたメッセージを残す。ところがしばしば、ユーザにはこのサブルーチンコールが並列処理に対して安全かどうかがわかっているばあいがある。その様なときには、cfpp$ではじまる指示文でこの情報をfppに伝えることによりfppは並列処理の手続きを踏

オリジナルコード

```
CFPP$ CNCALL
      DO 80 I = 1, N
         CALL CRUNCH (A(I), B(I))
80    CONTINUE
```

変換後

```
CMIC$ DO ALL
      DO 80 I = 1, N
         CALL CRUNCH( A(I), B(I))
80    CONTINUE
```

図 6. ユーザチューニング例

むことができるようになる。　図６の例では、

CFPP$　CNCALL

の指示文で　ループ内のサブルーチンコールが依存性の無いことを知らせることにより、fppは並列処理のための指示文

CMIC$　DOALL

を置いている。このように、オートタスキングにおいても、ユーザのコードに対する知識を指示文を通じてフィードバックしてやることによって更に効率の高いコードを作ることができる場合がある（ユーザチューニング）。図７に NASA で行ったオートタスキングの結果[2)]を示すがこのようなユーザチューニングによって非常に大きなスピードアップが得られる場合(Case 2、3、6、7)と、オートタスキングのみによって既に限界近くまで並列化の行われる場合(Case 1, 4, 5)とがあるのが分かる。

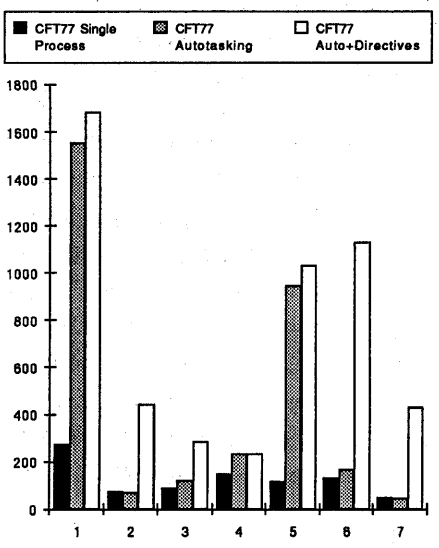


図 7. オートタスキングベンチマーク（ＮＡＳプロジェクト:ＮＡＳＡ）

fppの機能として大事なものに、ベクトル化促進を核としたFORTRANコードの最適化機能とサブルーチンおよび関数プログラムのインライン展開機能がある。

一般に並列化処理では高々使用可能なCPU数までのスピードアップしか期待できないのに対し、ベクトル化によるスピードアップはふつう数１０倍のオーダであり、並列化を行う時にも常にベクトル化を優先させなければならない。fppにおいても、先ずループの繰り返し、ストライド、依存性の割合、条件付き実行の割合等を解析した上でベクトル化に対する最適化を行い、その後で並列化の可能性を追求するという手順が踏まれている。従って、並列処理を行わない場合でもfppによって効率

オリジナルコード

```
      SUBROUTINE POTNTL(A,B,IP1,N)
      REAL A(*),B(*)
      DO 10 I = 1,N   ( 依存性の可能性あり)
        A(IP1+I)=A(I)+B(I)
10    CONTINUE
```

変換後

```
      IF(IP1.LE.0 .OR. IP1.GE.N) THEN
CDIR$ IVDEP
        DO 10 I = 1,N   ( ベクトルコード)
          A(IP1+I) = A(I) + B(I)
10      CONTINUE
      ELSE
CDIR$ NOVECTOR
        DO 77001 I = 1,N ( スカラーコード)
          A(IP1+I) = A(I) + B(I)
77001   CONTINUE
CDIR$ VECTOR
      ENDIF
```

図 8 - a. 条件付ベクトル化

の高いベクトル化コードを作ることができる。最適化技法としては、従来の自動ベクトル化コンパイラが持っているものはすべて含んだ上でさらに新しいものが追加されているので、それらを幾つか以下に紹介しておく。まず依存性の有無が実行時にしか分からない場合、スカラー、ベクトル両方のコードを生成し、実行時に　IF 文で使い分けるもので図８－ａにその例を示す。この場合 IP1の値いかんによって、このDO 10 ループは依存性を持ったり持たなかったりするが、ジョブの実行以前にその値が不定の場合、fppはこの例のような書換えを行う（条件付きベクトル化）。次の例は、図８－ｂにあるように、一つのループ

オリジナルコード

```
      DO 2530 L = 2, NLAYM1 （ スカラーループ）
         SDPOL(L,M) = SDPOL(L-1,M)+CONVPL(L)-
     1                DSIG(L)*PITPOL(M)
      IF(COMG) OMEGA(1,JKP,L) = SDPOL(L,M)*1.E6
2530  CONTINUE
```

変換後

```
CDIR$ IVDEP
      DO 2530 L = 1, J2S （ ベクトルループ）
         R1V(L) = CONVPL(1+J1S+L) -
     1        DSIG(1+J1S+L)*PITPOL(M)
2530  CONTINUE
      DO 77002 L = 1,J2S （ スカラーループ）
      SDPOL(1+J1S+L,M) = SDPOL(J1S+L,M)+R1V(L)
77002 CONTINUE
CDIR$ IVDEP
      DO 77003 L = 1, J2S （ ベクトルループ）
         IF(COMG) OMEGA(1,JKP,1+J1S+L) =
     1  SDPOL(1+J1S+L,M)*1.E6
77003 CONTINUE
```

図 8 - b. ループ分割

内に、依存性のある部分と、無い部分が混在するような時であるが、fppは、それらを分割して依存性の無い部分に対しては、ベクトルコードを生成する（ループ分割）。最後の例として、行列のかけ算を自動認識してクレイの高速ライブラリーに置き換える機能を図8－cにあげておく。クレイ・ライブラリーへの置き換えが行われるものとしては、行列積の他にも最大・最小値演算、あるいは一次、二次の線形依存性ループなどがある。

オリジナルコード

```
      PARAMETER ( N = 100 )
      COMMON A(N,N), B(N,N), C(N,N)
      ...
      DO 1 I = 1, N
         DO 11 J = 1, N
            A(I,J) = 0.0
11       CONTINUE
         DO 1 K = 1, N
            A(I,J) = A(I,J) + B(I,K)*C(K,J)
1     CONTINUE
      ...
      END
```

変換後

```
      CALL MXM (B, 100, C, 100, A, 100)
```

図 8 - c. ライブラリーによる置換

fppは、サブルーチンあるいは関数副プログラムのコーリングプログラムへの展開を行う。図9にその一例を示すが、このようなインライン展開はサブルーチンコールのオーバーヘッドを減らすため、しばしばアプリケーションコードに見られるように小さなサブルーチンや関数福プログラムが数多く呼ばれるような場合には非常に効果を発揮する。さらにインライン展開することにより、展開されたものがベクトル化や並列化さらにはスカラー最適化の対象ともなるため、一層効率のよいコードを作ることができる。

オリジナルコード

```
      DO 100 I = 1, N
         A(I) = CALC(A(I), X+B(I), 2.0)
100   CONTINUE
      ...
      END
      ...
      FUNCTION CALC( A, B, C)
         CALC = A + SQRT( B**2 + C**2)
         IF(CALC.LT.0) CALC = ABS( B + C )
      END
```

インライン展開後

```
      DO 100 I = 1, N
         TM1X = X + B(I)
         CALC1X = A(I)+SQRT(TM1X**2+2.0**2)
         IF ( CALC1X.LT.0 ) CALC1X =
     1                  ABS ( TM1X + 2.0 )
         A(I) = CALC1X
100   CONTINUE
```

図 9. インライン展開

4.1.2 コード変換ルーチン(fmp)

コード変換ルーチンfmpは、fppあるいはユーザによって入れられた指示文に従ってFORTRANソースコードを並列処理用コードに変換する役割をはたす。従って、fmpからは、並列処理に必要な様々な機械語ライブラリーへのコール文が埋め込まれた FORTRANコードが出力される。

4.1.3 コード生成ルーチン(cft77)

コード生成ルーチンは、上述したコード解析ルーチン、コード変換ルーチン等によって必要な修正、変換を施されたFORTRANコードを実行形式コードに変換するもので、現在前にも述べた cft77 Release 3.0が使われている。

4.2 オートタスキング使用例

本講演においては、オートタスキングの使用例をユーザチューニングの実際なども交えていくつか紹介するつもりであるが、その一例として図10に CRAY Y-MP (8cpu 32MW メインメモリー) を使って行った名古屋大学数学ライブラリー (NUMPAC[3]) のベンチマーク結果を示す。このベンチマークでは、今回説明したユーザチューニングなしに、Y-MP の8つのCPUを有効に使うコードが生成されている。

CHOLFWの速度性能 (msec)

OBS	N	VP4	S820	SX2N	Y-MP8
1	100	5	3	2	4
2	200	17	9	8	13
3	300	38	21	20	21
4	400	70	38	41	39
5	500	118	62	72	57
6	600	184	96	116	83
7	700	273	146	175	121
8	800	388	211	251	164
9	900	532	289	349	211
10	1000	708	382	467	260

HQR11Wの速度性能

全固有値 (msec)

OBS	N	VP4	S820	SX2N	Y-MP8
1	100	29	14	16	14
2	200	105	53	64	51
3	300	233	121	172	83
4	400	426	224	353	155
5	500	691	370	613	263
6	600	1046	576	1017	361
7	700	1498	838	1528	558
8	800	2063	1164	2198	898
9	900	2752	1558	3057	1022
10	1000	3564	2036	4072	1283

図10. NUMPACベンチマーク

5. まとめ

クレイ・リサーチ社のマルチタスキングについて、昨年発表されたオートタスキングを中心に解説した。FORTRANプログラムを入力として複数のCPUで並列処理できるコードを自動的に生成するという点でユーザにとって従来のマルチタスキング手法に比べ格段に使い易いものとなっている。

オートタスキングコンパイラシステムには、従来の自動ベクトル化機能に加えて新しくいくつかのベクトル化促進機能が追加されており、それらについても簡単な解説を行った。また、システムのオーバーヘッドを減らし、ベクトル化のみならずスカラーコードや並列化に対してもより効率の良いコードをつくり出す助けとなるサブプログラムのインライン展開についても、簡単な紹介を行った。

参考文献

1)加藤、大吉、三上:CRAY X-MP および CRAY-2 における大規模数値計算, スーパーコンピュータのための数値計算アルゴリズムの研究,数値解析研究所講究録613,京都大学数理解析研究所, pp.44-59, 1987年3月

2)Autotasking: automatic parallel processing on Cray systems, CRAY CHANNEL, Vol.10, No.4, A CRAY RESEARCH INC. PUBLICATION, WINTER 1989.

3)二宮、秦野:NUMPAC ﾍﾞｸﾄﾙｺﾝﾋﾟｭｰﾀの性能比較, SUPERCOMPUTER WORKSHOP REPORT 6, 分子科学研究所電子計算機センター, AUGUST,1988.