2007－ARC－174 (27)
2007／8／2

並列処理環境における消費電力量低減化方式の評価

佐藤裕幸[1]、尾崎敦夫[1]
[1]三菱電機（株）情報技術総合研究所

我々は並列処理環境の長時間連続運転による大量な消費電力、発熱の問題を解決することを目的に、動作周波数及び電圧が可変である省電力型プロセッサを利用して、制限時間までに間に合いかつ最も省電力化できるよう動作周波数及びプロセッサ数を選定する方式を提案してきた。今回、この実行方式の有効性を検証するために、我々が対象とする代表的なアプリケーションである多目標追尾処理プログラムを用いて、評価実験を行った。評価実験の結果、現在市販されている単一プロセッサ用の省電力化方式に比べて6割～8割、省電力化を行わない方式に比べて2割～4割程度の消費電力量になり、稼働率が低いほど効果が高くなることを確認した。

Evaluation of CPU Power Control and Scheduling Technique on Parallel Computing Environment

Hiroyuki Sato[1]、 Atsuo Ozaki[1]、
[1]Mitsubishi Electric Corp. Information Technology R&D Center

A power consumption and thermal ascent problems are serious problem when the system is used for a long time. We had proposed CPU power control and scheduling technique (PCST) on parallel and distributed computing environment, in which the processors can be changed these frequencies and voltages, to solve these problems. In this paper, we evaluate the PCST using the tracking program, which is our typical application. Evaluation results show that the power consumption on PCST is 60% - 80% compared with that on the commercial power control, 20% - 40% compared with that without the power control.

1 はじめに

処理の高速化に対する要望は留まることを知らず、並列処理環境も大規模化が進んでいる。しかしこの大規模化に伴い、プロセッサ数が多くなると、消費電力、発熱、そして設置スペースの問題などが新たに出てくる。特に、長時間連続運用するシステムでは、消費電力及び熱の問題が深刻となる。

一方、モバイルコンピューティングの分野では、バッテリ駆動により長時間使用できる技術が必須となる。近年、消費電力量削減のために、必要に応じて動作周波数及びコア電圧を切り替えることが可能なプロセッサが登場し、高性能な製品群にまで展開され始めてきている。

我々が対象としているアプリケーションの特徴は、システムに周期時間毎に入力されるデータをこの周期時間内に処理していくということである。この周期時間毎に入力されるデータ数及びデータ量は、その都度変化し、処理負荷も変動する。これまで、この処理を行うシステム（並列処理環境）では、該周期時間に最も多くのデータ数及びデータ量が入力される事態を想定してプロセッサ数を設定していた。そのため、通常はほとんどのプロセッサが遊休状態であり、無駄な多くの電力量を消費しているのが実情である。

そこで我々は、並列処理環境の省電力化の研究開発を実施し、その成果として、処理の負荷に応じた、使用プロセッサ数と動作周波数及びコア電圧を選定する基準（省電力化のための実行方式）を考案した[1]。今回、この実行方式の有効性を検証するために、我々が対象とする代表的なアプリケーションである多目標追尾処理プログラムを用いて、動作周波数及びコア電圧を切り替え可能なプロセッサから成る並列処理環境上で、評価実験を行ったので報告する。

まず、考案した省電力化のための実行方式について述べ、次に対象とする並列追尾処理プログラムについて説明し、その後、評価実験結果について報告する。

2　消費電力量低減のための実行方式

2.1　基本概念

現在主流の CMOS 回路では、基本的に動作周波数を上げた場合にはプロセッサを安定動作させるためにコア電圧も上げる必要がある。逆に、動作周波数を下げた時にはコア電圧も下げることが可能な場合がある。一般的に、消費電力 P[W]と動作周波数 F 及びコア電圧 V との関係は、リーク電力を無視した場合、式(1)となる。ここで、S は信号遷移率、C は静電容量である。

$$P = S \cdot C \cdot F \cdot V^2 \qquad (1)$$

消費電力 P は、式(1)に示すように、動作周波数 F に比例し、かつコア電圧 V の二乗に比例するという両要素が支配的である。すなわち、消費電力 P は、動作周波数 F の変化率に対して、三乗のオーダで推移する。また、消費電力量は時間を掛けた P×t [W・s]であり、実行時間 t は動作周波数 F に反比例する。このため、まずは対象とする並列処理環境の遊休状態であるプロセッサの動作周波数及びコア電圧を設定可能な最低値にすることで、大幅な消費電力量 [W・s]の削減が実現できるが、我々は更に、タスクの実行方式を工夫することによる更なる消費電力量の低減化を考案した。

我々が対象としているアプリケーションは、制限時間（次の周期時間）内に処理を完了すれば良い。従って、処理負荷に余裕がある場合は、制限時間内に処理が完了するよう、動作周波数を低めることで、消費電力量を削減できる。更に並列処理環境においては、動作周波数だけではなく、使用するプロセッサを減らすことでも同様の効果が期待できる。従って、制限時間内に処理を完了させるために必要な演算能力を得るために、高クロックで少数プロセッサで実行するのか、低クロックで多数プロセッサで実行するのかを選定する。すなわち、動作周波数とプロセッサ数の２つのパラメータを調整することで、消費電力を削減する実行方式である。

2.2　実行方式選定手法

我々が提案した実行方式選定手法は、タスク（処理データ）が与えられる度に、与えられた処理制限時間内に最も少ない消費電力量で処理することができるプロセッサ数とそれらのプロセッサの動作周波数及びコア電圧を選定するものである。なお、使用していないプロセッサ及び処理していないプロセッサは、最も電力を消費しない動作周波数及びコア電圧に設定する。

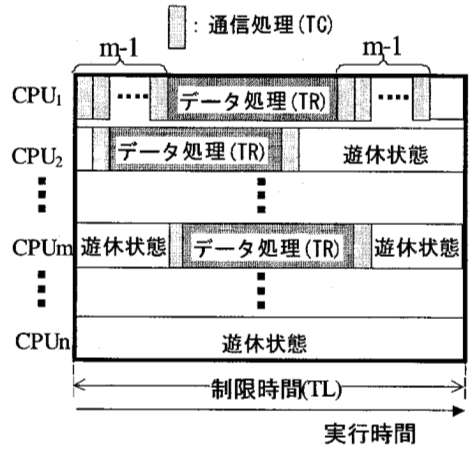

図１　タイムチャート

図１は、n 個のプロセッサの内、m 個のプロセッサを用いた典型的な並列処理のタイムチャートの例である。マスターとなる CPU1 は、自分以外の m-1 個のプロセッサに対して、データ処理を行うためのデータを送信した後、データ処理を行い、m-1 個のプロセッサから処理結果を受信し、それが終ったら遊休状態に入る。データ処理を受け持つ CPU2～CPUm は、CPU1 からのデータを受信するまで遊休状態であり、データを受信したらデータ処理を行い、処理結果を CPU1 に送信する。使用しない CPUm+1～CPUn は、常に遊休状態である。

並列処理により、各プロセッサ（CPU1～CPUm）のデータ処理時間及び送受信時間が等しいと仮定すると、このような並列処理における全体の消費電力量は、以下のように算出できる。

m・TR・P ＋	// 使用 CPU の実行中
(m-1)・TC・P ＋	// CPU1 の通信中
(m-1)・TC・P ＋	// CPU2～CPUm の通信中
(TL-TR-(m-1)・TC)・PI ＋	// CPU1 の遊休中
(m－1)・(TL-TR-TC)・PI ＋	// CPU2～CPUm の遊休中
(n-m)・TL・PI	// CPUm+1～CPUn の遊休中

n：全プロセッサ数、m：使用プロセッサ数、
TR：各 CPU のデータ処理時間、
TC：各 CPU の通信時間、TL：制限時間
P：当該動作周波数での消費電力、
PI：最低動作周波数での消費電力

各動作周波数 P_i での処理時間、通信時間を見積もり、各動作周波数での制限時間を守れるために必要な最

少プロセッサ数を求め、それぞれの消費電力量を上記の式で算出し、その結果が最も少ない動作周波数及びプロセッサ数で実行する[1]。通常、利用可能な動作周波数は数種類であり、制限時間を守れる必要最少プロセッサ数を求める手間もそれほど大きくないことから、この実行方式選定の手間も大きくないと考えられる。

3 評価対象プログラム

この省電力のための実行方式選定手法の有効性を検証するために、センサー処理で代表的なアプリケーションである並列航跡型 MHT (Multiple Hypothesis Tracking) [2] [3] を用いて評価実験を行った。MHT は、多目標追尾処理を行うプログラムであり、その主な処理は、これまでの追尾結果である航跡（これまで探知されたプロットデータの内、同一目標として判断した探知データを時系列に連結したもの）の現時刻における予測範囲（ゲートと呼ぶ）と現時刻に探知されたプロットデータの相関を取ることである（図２）。複数の航跡のゲートが重なり合う場合には、複数の航跡が同一の探知データに対応する可能性もあるため、同時に成立し合わない組み合わせを仮説として複数保持しながら処理を進めていく。また、センサーの能力により、目標がいるのに探知されない場合や、目標がいないのに探知される場合があるため、航跡と探知データの組み合わせ数が多くなるため、相関処理はかなりの高負荷となる。

各航跡の相関処理は、その航跡のゲート内の探知データ数（相関探知データ数）に依存する。従って、

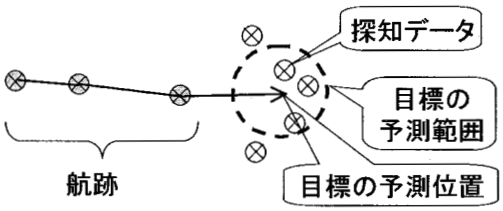

図２ 航跡と探知データの相関処理

[1] [1] では、更に、アプリケーションの実際の実行時間が不明確であっても、データ処理時間に対する通信処理時間の比率が分かれば、省電力のための実行方式を選定可能としていた。本評価で対象とするアプリケーションは実行時間が見積もれるため、通信比率による選定は行わない。

全体の処理負荷は、航跡数と各航跡の相関探知データ数に依存する。

3.1 並列化手法

この航跡型 MHT にはさまざまなレベルで並列性がある[4] が、今回はゲートが重なっている航跡の集合（クラスタと呼ぶ）単位で並列化を行った。個々のクラスタでの追尾処理は、それぞれ独立して処理可能なので、並列化が比較的容易に行える。これはほぼ追尾領域単位での並列処理になるが、個々の領域のサイズ（並列粒度）は、追尾状況によって変化する。

並列化対象のクラスタの処理負荷はある程度見積もれて、更にクラスタの数は、使用するプロセッサの数に比べてそれほど多くないということもあり、今回は実行前にどのクラスタをどのプロセッサに担当させるかを決める静的負荷分散方式を採用した。まず、各クラスタについて、それぞれ「航跡数×相関探知データ数」を算出し、これをクラスタの負荷予測値とする。そして、負荷予測値の大きいクラスタから順に、１つずつ各プロセッサに割付けていき、全てのプロセッサに割り付けたら、また最初のプロセッサから割り付けていく（サイクリック割付）。

また、各プロセッサで、相関処理を行うために、マスタープロセッサからクラスタ・データ（クラスタ内の仮説、航跡、フィルタ情報など）を担当するプロセッサに送信し、処理完了後、マスタープロセッサに更新されたクラスタ・データを返す。このクラスタ・データは、かなりの割合を航跡データが占めているので、その通信時間は、航跡数に依存すると考えられる。なお、非常に負荷が低いと予測されるクラスタに関しては、通信オーバヘッドが負荷に比べて相対的に大きくなるため、他プロセッサに割り付けず、マスタープロセッサで処理する。

3.2 処理時間の見積り

各プロセッサでの負荷は、「航跡数×相関探知データ数」に依存すると考えたが、実際の処理時間は航跡の状況に応じて異なっている。すなわち、複数の航跡が並行している状況では高々2 つの航跡が同じ探知データと相関がある程度で負荷が低く、複数の航跡が交錯している状況では多くの航跡が同じ探知データと相関があり負荷が高いと考えられる。

18 個の目標が互いに交差する状況で 2,000 探知時刻分の各クラスタの処理時間を計測し、負荷予測値（航跡数×相関探知データ数）とその処理時間（単位：ミ

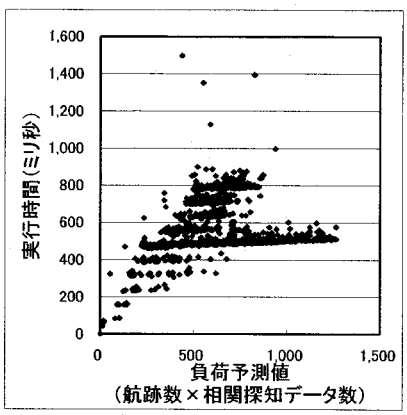

図３ 負荷予測値と実行時間（2,000 探知時刻）

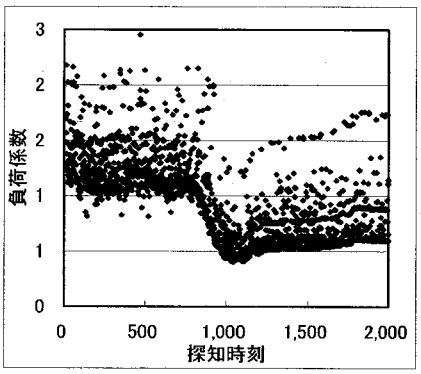

図４ 探知時刻毎の負荷係数の推移

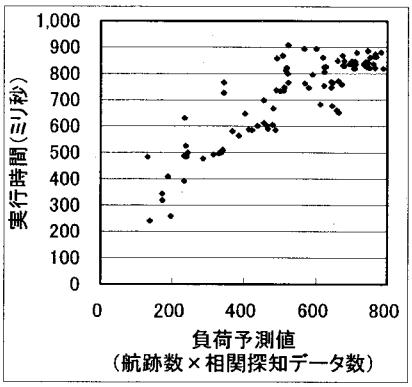

図５ 負荷予測値と実行時間（100 探知時刻）

リ秒）をプロットしたものを図３に示す。各追尾目標同士は、それぞれ離れて行ったり近づいて行ったりしており、時刻と共に状況が変化していくため、負荷の予測値と実際の実行時間の関係は、図３に示す通り相

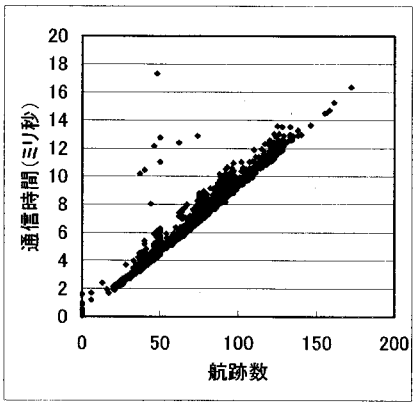

図６ 航跡数と通信時間

関がないように思える。すなわち、同じ負荷予測値でも実際の処理時間が異なっていた。

ここで、1負荷予測値当たりの実行時間、すなわち、実行時間を負荷予測値で割った値を負荷係数と呼ぶことにする。この負荷係数が常に一定であれば、負荷予測値から実行時間を見積もることが可能である。この負荷係数を探知時刻毎にプロットすると、図４のようになる。この負荷係数の推移を見ると、個々の探知時刻では、ある程度幅はあるが一定の負荷係数となっており、それが時刻と共に緩やかに変化している事が分かる。図５に100探知時刻分の負荷予測値と実行時間のプロットを示す。この短い時刻間でプロットすると、負荷予測値と実行時間には相関があり、負荷係数もある程度一定になっている。

従って、各クラスタの処理時間を計測し、負荷予測値（航跡数×相関探知データ数）から負荷係数を求め、それをある一定の探知時刻間で平均することで、負荷予測値から処理時間を見積もることが可能である。

3.3 通信時間の見積り

通信時間は、航跡数に依存すると考えたが、航跡数と実際の通信時間（単位：ミリ秒）をプロットしたものを図６に示す。通信時間に関しては、追尾状況とは無関係なので、ある程度の計測誤差はあるものの、航跡数と通信時間の関係は一定であり、航跡数から通信時間を見積もることが可能である。

4 計測評価

２章で示した省電力化のための実行方式を検証するために、３章で示した並列航跡型 MHT プログラムを用いて、動作周波数及びコア電圧を切り替え可能な

表１　実験環境

項目	仕様
PC	イーレッツ BeSilent Mt6600
CPU	Transmeta Efficeon TM8600 1GHz
メモリ	DDR-SDRAM 512MB
ネットワーク	Gigabit Ethernet(RTL8110s)
OS	Linux kernel 2.4.22
コンパイラー	gcc 3.3.2
通信ライブラリー	PVM 3.4

表２ Efficeon の電力モードと消費電力

電力モード	動作周波数 (MHz)	コア電圧 (V)	消費電流 (A)	消費電力 (W)
0	533	0.850	1.30	1.11
1	600	0.925	1.58	1.46
2	700	1.050	2.09	2.19
3	833	1.100	2.70	3.11
4	1,000	1.300	3.73	4.85

プロセッサから成る並列処理環境上で、評価実験を行った。

4.1　実験環境

今回の実験では、トランスメタ社の Efficeon プロセッサを搭載した 4 台の PC から成るクラスタを使用した。その構成は、表１の通りである。

この Efficeon TM8600 プロセッサは、5 つの動作周波数及びコア電圧（電力モード）を設定でき、（逐次の）航跡型 MHT プログラムによる計測により、それぞれの電力モードにおける消費電力は表２の通りになった。今回の評価実験での消費電力量[W・ms]の計測は、計測器による実測ではなく、各電力モードでの実行時間をログに取り、表２の各電力モードにおける消費電力にその電力モードでの実行時間を乗じた値の総和で、消費電力量とした。

4.2　比較のためのその他の実行方式

我々の考案した省電力化の実行方式の有効性を検証するために、全く同じ条件で、以下のその他の実行方式を採用した場合の消費電力量の計測も行った。以下に今回計測した実行方式を示す。

(1) 提案方式

我々が考案した、制限時間に間に合いかつ消費電力量が最小となるよう、プロセッサ（CPU）数と電力モード（動作周波数及びコア電圧）を設定して実行する。

(2) 最小プロセッサ方式

制限時間に間に合いかつ使用プロセッサ数が最少となるよう、プロセッサ数と電力モードを設定して実行する。本方式は、制限時間に間に合うよう、必要最小限のプロセッサのみ使用するという意味で、省電力のための最も自然に考えられる実行方式である。

(3) 従来省電力化方式

最小時間で実行できるよう最多プロセッサを用い、データ処理中は最高動作周波数（電力モード 4）で、処理完了後及び通信待ち時間中は最低動作周波数（電力モード 0）で実行する。本実行方式は、電力制御可能なチップメーカであるインテル社やトランスメタ社が提供している従来の省電力化のための実行方式である。

(4) 非省電力化方式

最小時間で実行できるよう最多プロセッサを用い、常に最高動作周波数（電力モード 4）で実行する。すなわち、電力制御を全く行わない。

上記の 4 つの実行方式について、全く同じ探知データを用いて、1 探知処理当たりの制限時間が 0.6 秒、1.2 秒、2.4 秒の場合について計測した。この制限時間が長いほど、プロセッサの稼働率（プロセッサが何らかの処理をしている割合）が低くなり、電力制御の効果が高くなるはずである。なお、今回用いた探知データでは、制限時間 0.6 秒の場合が最大負荷であり、制限時間をそれより短くすると、最高動作周波数、最多プロセッサを用いても制限時間に間に合わなくなってしまう。

4.3　計測結果

上記の実行方式(1)～(4)の 3 つの制限時間に対する消費電力量（単位：W・ms）とそれぞれの稼働率を表３に示す。

消費電力量は、制限時間までに間に合うように動作周波数を落として実行する(1)、(2)が(3)、(4)に比べて明らかに低く、動作周波数を落としたことによる効果が鮮明に出ている。更に、提案方式(1)は、省電力のための最も自然に考えられる実行方式である(2)に比べて、制限時間 2.4 秒を除いて明確な差が出ており、提案方式の有効性を確認できた。なお、制限時間 2.4 秒の場合は、許される実行時間に余裕があるため、別途記録した実行ログからも、ほとんどの実行が 1 プロセッサで最低動作周波数での実行となっており、(2)と結果的にほとんど同じ実行方式となっているため、消費電力量に差が出なかった。

提案方式(1)は、現在市販されている単一プロセッサ用の省電力化方式(3)に比べて、高負荷時で 8 割弱、低負荷時で 6 割弱の消費電力量となっており、高い省電力化効果が現れている。また、省電力制御をしていな

表 3 計測結果

制約時間：0.6秒

	稼働率	(1)	(2)	(3)	(4)
CPU1	55.07%	134,670	157,851	195,767	299,155
CPU2	41.03%	137,286	151,515	162,369	297,287
CPU3	37.93%	131,977	137,937	155,081	296,849
CPU4	35.57%	124,677	122,395	149,487	296,574
平均/合計	42.40%	528,610	569,697	662,704	1,189,865
(1)に対する割合		100.00%	107.77%	125.37%	225.09%
(2)に対する割合		92.79%	100.00%	116.33%	208.86%
(3)に対する割合		79.77%	85.97%	100.00%	179.55%
(4)に対する割合		44.43%	47.88%	55.70%	100.00%

制約時間：1.2秒

	稼働率	(1)	(2)	(3)	(4)
CPU1	27.65%	135,499	210,176	261,094	589,912
CPU2	20.62%	134,775	170,202	227,205	585,124
CPU3	18.94%	134,663	134,655	219,510	584,784
CPU4	17.59%	134,086	134,543	213,264	584,408
合計	21.20%	539,023	649,576	921,074	2,344,228
(1)に対する割合		100.00%	120.51%	170.88%	434.90%
(2)に対する割合		82.98%	100.00%	141.80%	360.89%
(3)に対する割合		58.52%	70.52%	100.00%	254.51%
(4)に対する割合		22.99%	27.71%	39.29%	100.00%

制約時間：2.4秒

	稼働率	(1)	(2)	(3)	(4)
CPU1	13.87%	268,778	269,235	394,586	1,173,148
CPU2	10.39%	266,224	266,271	360,247	1,162,604
CPU3	9.49%	266,125	266,173	352,681	1,162,258
CPU4	8.81%	266,010	266,061	347,370	1,161,861
合計	10.64%	1,067,138	1,067,739	1,454,884	4,659,871
(1)に対する割合		100.00%	100.06%	136.34%	436.67%
(2)に対する割合		99.94%	100.00%	136.26%	436.42%
(3)に対する割合		73.35%	73.39%	100.00%	320.29%
(4)に対する割合		22.90%	22.91%	31.22%	100.00%

い(4)と比べると、稼働率に近い消費電力量となっており、稼動していない時でも電力を消費していることを考えると、かなり高い効果が出ていると言える。今回、実験に使用したプロセッサ Efficeon は、元々低消費電力型のプロセッサであり、省電力化の効果が薄いと言われている中での結果としては、高い効果を確認することができた。

実行ログを見ると、提案方式(1)では、結果的に低い動作周波数で多くのプロセッサを使用している。これは、この実験環境のプロセッサの電力特性（動作周波数及びコア電圧と消費電力の関係）により、このような実行方式の方が省電力化できたということである。従って、異なる電力特性を持った環境では、高い動作周波数で少ないプロセッサの方が良くなる可能性もある。しかし、我々が提案した方式では、消費電力量を見積もるので、実行環境の電力特性に適した実行方式を選定できる。なお、今回実験に使用したプロセッサ Efficeon は、現時点では製造中止となり、今後使用する可能性はないが、我々が考案した実行方式は、プロセッサに依存した方式ではないので、他の電力制御可能なプロセッサでも高い効果が期待できる。

最後に、使用する電力モードとプロセッサ数を選定するための処理時間は、実行ログから、長くても数ミリ秒程度で、ほとんどが 200 マイクロ秒弱であり、全体の処理時間と比べると、全くオーバヘッドがなく選定できている。

5 おわりに

以上、我々が考案した省電力化のための実行方式の有効性を検証するための評価実験結果について報告した。

我々の考案した実行方式は、制限時間までに間に合いかつ最も省電力化できるよう動作周波数及びプロセッサ数を選定する方式である。評価実験の結果、現在市販されている単一プロセッサ用の省電力化方式に比べて 6 割～8 割、省電力化を行わない方式に比べて 2 割～4 割程度の消費電力量になり、稼働率が低いほど効果が高くなることを確認した。

我々の対象としているセンサー処理では、目標の状況により負荷（稼働率）の変動が激しく、従来の大規模センサー処理システムでも最大負荷を想定してプロセッサ数を設定しているため、かなり低い稼働率となっている。これらのシステムでは、周波数制御可能なプロセッサが利用できなかったが、現在は PowerPC 系のプロセッサでも周波数制御可能なものが供給され始めており、今後は利用可能になっていく。従って、センサー処理システムでも、今後はこれらのプロセッサを利用して我々の考案した実行方式を採用することにより、かなりの省電力化が図れると考えられる。

参考文献

[1] 尾崎敦夫, 佐藤裕幸：“並列処理環境における消費電力量低減化方式の検討, “ 情処研究報告, 2004-ARC-159, 2004.

[2] 小菅義夫, 辻道信吾, 立花康夫：”航跡型多重仮説相関方式を用いた多目標追尾, “信学論(B-II), vol. J79-B-II, no. 10, pp. 677-685, 1996.

[3] 小幡康, 系正義, 辻道信吾, 小菅義夫：”N ベスト解探索アルゴリズムによる航跡型 MHT の高速化(1) —N ベスト解探索アルゴリズムの導入方式—, “ 2001 信学総大, B-2-32, 2001.

[4] 高橋 勝己, 佐藤 裕幸：” N ベスト仮説の探索木分配による並列処理, “ 2003 信学総大, B-2-29, 2003.