Vol.2009-UBI-23 No.6
2009/7/16

無線センサノードにおける シングルコアCPUの問題点に関する定量的評価

大原 壮太郎[†1] 鈴木 誠[†1] 猿渡 俊介[†1]
南 正輝[†1] 森川 博之[†1]

本稿では，既存のシングルコアで構築されるセンサネットワークのアプリケーションを分析し，マルチコア化によってタスクスケジューラを不要とすることができればセンサノードの消費電力とタスクの実行遅延を大きく削減できることを示す．シングルコアのセンサノードにおけるスケジューラの消費電力やタスクの実行遅延の定量化を目的とし，実際のセンサノード上で複数のアプリケーションを動作させ，実測評価を行った．評価結果によれば，スケジューラに起因する電力消費とタスクの実行遅延がトレードオフの関係にあり，同時に解決することが困難であることが分かった．また，マルチコア化の実現可能性を検証するために，現在の無線センサネットワークのアプリケーションにおける，タスクフローの解析を行った．その結果，タスクフローの解析からマルチコア化した際に必要なコアの数を抽出可能であることが示された．

Quantitative Evaluations of Problems in Single-Core Wireless Sensor Nodes

Sotaro Ohara,[†1] Makoto Suzuki,[†1]
Shunsuke Saruwatari,[†1] Masateru Minami[†1]
and Hiroyuki Morikawa[†1]

This paper shows a multicore sensor node can eliminate a task scheduler, and reduces power consumption and task execution delay, which are problems in an existing single-core sensor node. To quantify the power consumption and the task execution delay, an experimental evaluation run various services on a wireless sensor node. The evaluation result shows that it is difficult to reduce power consumption and task execution delay at the same time because there is a trade-off between them. Additionally, to validate feasibility of the multicore sensor node, the task flow analysis on software in wireless sensor networks is conducted. The analysis result shows that we can estimate a number of required cores to implement the software on a multicore sensor node.

1. はじめに

無線センサネットワークは，センシング機能を有するコンピュータを無線通信によりネットワーク接続し，実空間の情報を収集する技術である．これまでに時刻同期やルーティングプロトコル，位置推定等，要素技術の研究が進み，近年になってアプリケーションの開発が盛んにおこなわれている．[1)2)]

無線センサネットワークのアプリケーションを開発する際，多くの場合 TinyOS とシングルコア CPU を具備するセンサノードが用いられている．TinyOS[3)14)] はイベント駆動型のオペレーティングシステムであり，イベントの発生により実行されるタスクを run-to-completion で実行する．TinyOS はイベントモデルに特化することによる省資源性や，個々の CPU に特有の機能を使用しないことによる高い移植性により，無線センサネットワークの標準的なオペレーティングシステムとして扱われている．また，センサノードは MICA mote ファミリーなど汎用のセンサノードが利用されており，これまでのところ，すべての汎用センサノードがシングルコアの CPU を具備している．

TinyOS とシングルコアのセンサノードでは，タスクスケジューリングに関して同時に解決することが困難な 2 つの問題が発生する．1 つ目の問題は，スケジューリングによる電力消費である．シングルコア CPU で動作するオペレーティングシステムでは，CPU 資源を複数のタスクに割り当ててタスク処理の並列性を確保するためにスケジューラが必要となる．TinyOS では，機能を FIFO（First In First Out）によるタスク実行に絞ることで，103 cycle という低オーバーヘッドのスケジューリングを実現している．しかし，無線センサノードの CPU で処理される多くのタスクはスケジューリングのオーバヘッドと同程度に計算量が小さく，103 cycle のオーバーヘッドを無視することはできない．例えば，MTS300[4)] 上のセンサの電源を TinyOS からオフにするタスクの処理時間は 160 cycle となる．

2 つ目の問題は，タスク実行の際に発生する遅延である．TinyOS のスケジューラは FIFO でタスクに CPU 資源を割り当てるため，タスクをタスクキューに挿入する際に他のタスクが実行中の場合にはタスク実行に遅延が発生する．タスクが非同期に発生する場合，遅延の大きさが不確定となるため，シングルコア CPU 上で駆動する TinyOS のスケジューラでは

†1 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

タスク処理の終了時刻を保証することができない．例えば，センサ情報を FFT している間に他のセンサノードからパケットが到着した場合，FFT の計算負荷によって生じる遅延によってパケットを受信できないことが起こりうる．

シングルコアのセンサノードでは，スケジューラのオーバーヘッドとタスク実行の遅延はトレードオフの関係にあるため，省電力性とリアルタイム性を同時に実現することが困難である．スケジューラのオーバーヘッドを軽減するためには，タスクの粒度を大きくしてスケジューリングの実行回数を減らせばよい．ところが，タスクの粒度を大きくすると 1 つのタスクが CPU を占有する時間が増加し，タスク処理の遅延も増大する．逆にタスクの粒度を小さくした場合，タスク処理の遅延は小さくなるものの，スケジューリングの実行回数が増え，スケジューラのオーバーヘッドが増加する．

このような問題に対し，筆者らはセンサノードの消費電力を削減すると共にハードリアルタイム処理を伴うソフトウェア開発の効率化を目指し，無線センサノード向けマルチコア CPU「Mulco」の研究開発を行っている．本稿では，Mulco の実現に先立ち，シングルコア CPU の問題点を明確化することを目的とする．具体的には，シングルコア CPU においてスケジューラが電力消費とタスクの実行遅延の 2 点に大きな影響を与えることを定量的評価によって示す．また，シングルコア CPU 上で動作するアプリケーションのタスクフローの解析を通じ，スケジューラの電力消費や実行遅延の問題をマルチコア CPU の利用により解決可能であることも明らかにする．

本稿の構成は以下の通りである．まず 2. で現在のシングルコア CPU のセンサノードにおける問題点を述べる．3. では，2. に示したシングルコア CPU のセンサノードにおける問題点の定量的評価の結果を示す．次いで，4. では，シングルコア CPU のセンサノード上で動作するアプリケーションのタスクフローを分析し，マルチコア CPU のセンサノードを用いることでシングルコア CPU での問題点が解決可能であることを示す．さらに，5. で無線センサネットワークに向けたマルチコア CPU 実現のための考察を行い，最後に 6. でまとめとする．

2. 無線センサネットワーク

2.1 現在のシステム構成

現在，無線センサネットワークのアプリケーションを構築する際，シングルコア CPU を具備したセンサノードと TinyOS の組み合わせが最も標準的に使用される．

センサノードは主に CPU，センサ，無線モジュールから構成される．汎用センサノードは，市販されている電子部品を組み合わせて実現されており，汎用性と消費電力等のバランスを考慮して設計されている．代表的な汎用センサノードとそれぞれが具備する CPU を表 1 に示す．表 1 から分かるように，汎用性と消費電力の観点から，多くの汎用センサノードがシングルコアの 8 bit CPU を具備している．汎用センサノードでアプリケーションの要件を満たせない場合は，専用のセンサノードを設計してアプリケーションを構築する．例えば，Park らの構造モニタリングの研究では[7]，サンプリングのリアルタイム性を確保するために 2 つの PIC18 マイコンを具備したセンサノードを構築している．

表 1 汎用のセンサノードと具備している CPU

Node	Developing Group	CPU
Micaz	UC Berkeley (US)	ATMEL
BT node	ETH Zurich (Swiss)	Atmega128
TerosB	UC Berkeley (US)	Texas Instruments
BSN node	Imperial Collage London (UK)	MSP430
PAVENET	University of Tokyo (Japan)	Microchip PIC18F4620

シングルコアのセンサノード上でアプリケーションを処理する場合，オペレーティングシステムは，CPU 資源を 1 つの処理が独占しないように分配し，処理の並列性を確保する必要がある．センサノード向けのオペレーティングシステムとして TinyOS が挙げられる．TinyOS は，カリフォルニア大学バークレー校を中心とするグループにより開発された無線センサネットワーク向けのオペレーティングシステムであり，省電力性，省資源性，移植性の高さから多くの支持を得ている．

TinyOS において，デフォルトのスケジューラである SchedulerBasicP[15] を用いてタスクを実行する様子を図 1 に示す．TinyOS では，`event` と `task` の 2 つの処理形式が用意されており，通常の処理を `task` として実装することで並列処理を実現している．`event` はハードウェア割り込み，他の `event`，`task` から呼び出されて実行され，割り込みから呼び出された場合は `task` を preemption できる．`task` は `event` や他の `task` から実行要求を受けてスケジューラを通して non-preemptive に実行される．具体的には，CPU はタイマや外部機器からのハードウェア割り込みを受けた時，対応する `event` が実行され，必要であればその `event` 内で `task` をタスクキューに挿入する（`post`）．最後にスケジューラがタスクキューから FIFO で `task` を取り出し，順番に実行（`run`）する．スケジューラは，タスクキューに実行待ちのタスクが存在しなくなった場合，CPU をスリープさせる．SchedulerBasicP では `task` を `post` および `run` する処理に 103 cycle を要する．また，タスクキューに実行待ちの `task` が存在しないことを確認して CPU をスリープさせる処理に 20 cycle を要する．MicaZ[5] の駆動周波数である 7.37 MHz ではそれぞれ 14.0 μs，2.7 μs に相当する．

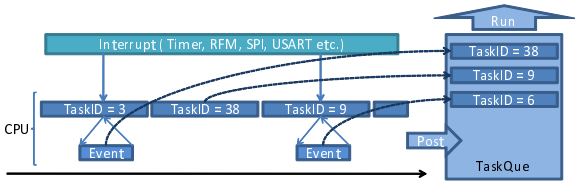

図 1 TinyOS のタスク実行の様子

2.2 シングルコアセンサノードの問題点

シングルコア CPU と TinyOS の組み合わせでアプリケーションを処理すると，同時に解決することが困難な 2 つの問題が発生する．1 つ目の問題は，スケジューリングによる電力消費である．無線センサノードの CPU で処理されるタスクの多くは計算量が小さく，103 cycle というタスクスケジューラのオーバーヘッドを無視できない．Shnayder らは，汎用センサノードである Mica2 において，実際のアプリケーションを実行した際の CPU，無線モジュールおよびセンサ等の構成要素ごとの消費電力を測定し，CPU での消費電力はノード全体の消費電力の 28 % から 86 %，平均で約 50 % を占めることを示している[8]．すなわち，CPU で実行されるスケジューラのオーバーヘッドが大きい場合，その電力消費のノード全体の消費電力への影響は大きい．

2 つ目の問題は，タスク実行の際に発生する遅延である．遅延の発生は以下の 2 つに因る．1 つ目の遅延要因は，スケジューラ自身の処理によって発生する 14.0μs の遅延である．この遅延はすべてのタスクに対して等しく発生する．2 つ目の遅延要因は，あるタスクを `post` する際に実行中のタスク及びタスクキューに入っているタスクの処理によって発生する遅延である．2 つ目の遅延は先発タスクによって決定されるため，不確定な大きさの遅延となる．これらの遅延の存在は，時間制約の厳しいタスクを処理する際に問題となる．無線通信や正確な周期でのサンプリング等，時間制約の厳しいタスクを多く含む無線センサネットワークでは，タスク処理の時間制約を満たすリアルタイム処理を保証することが必要となる．例えば橋梁に加速度センサを設置し，揺れを観測する研究[9] では，TinyOS のスケジューラにより生じる不確定な大きさの遅延によって，サンプリング値の有用性が著しく損なわれることが報告されている．

現在のシングルコア CPU のシステム上でスケジューリングによる電力消費とタスクの実行遅延の発生という問題を同時に解決し，省電力での動作とリアルタイム保証を同時に実現することは困難である．スケジューリングのオーバーヘッドを軽減するためには，分割されて実装されているタスクをまとめて 1 つのタスクとし，一度のスケジューリングで多くの処理を行うことが考えられる．しかし，1 つのタスクの計算量が大きくなると，前発タスクの処理による後発タスクの実行遅延が増大する．つまり，2 つの問題はお互いにトレードオフの関係にあり現在のシングルコア CPU のシステムでは同時に解決することが困難である．

```
command void Scheduler.taskLoop(){
  for (;;){
    uint8_t nextTask;
    atomic{
      while ((nextTask = popTask() == NO_TASK){ … (1)
        call McuSleep.sleep(); … (2)
      }
    }
    setTaskIDStart(nextTask); … (3)
    signal TaskBasic.runTask[nextTask](); … (4)
    setTaskIDStop(nextTask); … (5)
  }
}
```

図 2 SchedulerBasicP

3. 電力消費と遅延の定量的評価

前節までの議論を踏まえ，本節では現在のシングルコア CPU における問題点を明確にするために，タスクスケジューラに関する省電力性と遅延の問題について定量的な評価を行う．

3.1 スケジューリングによる電力消費

スケジューリングによる CPU の電力消費の測定のために，TinyOS の複数のサンプルアプリケーションにおけるタスクの実行要求／実行開始／実行終了のタイミング，割り込みルーチンの実行開始／実行終了のタイミングを記録した．具体的には，センサノードとして MicaZ[5] を利用し，実行開始時および実行終了時に汎用 I/O にタスク ID，割り込み ID を出力する測定用のコードをスケジューラおよび割り込みルーチンに挿入する．サンプリング周波数が 100 MHz のロジックアナライザを利用してタスクおよび割り込みごとに実行の開始および終了のタイミングを記録した．

例として，図 2 に TinyOS のデフォルトのスケジューラである SchedulerBasicP のソースコードと測定用コードの挿入部を示す．SchedulerBasicP では，(1) でタスクキューの先頭を取り出し，タスクキューに実行待ちのタスクが存在しない場合は (2) で CPU をスリー

表 2 Sense のタスク

TaskID	Task	Number of Times	Cost (μs)
0	AlarmToTimerC/0/fired	194	59.1
1	VirtualizeTimerC/0/updateFromTimer	291	57.0
2	ArbiterP/0/grantedTask	97	37.1
3	ArbiterP/1/grantedTask	97	30.0
4	ArbiterP/2/grantedTask	0	—
5	PowerManagerP/0/start	97	31.3
6	PowerManagerP/0/stop	97	21.7
7	PowerManagerP/1/startTask	0	—
8	PowerManagerP/1/stopTask	0	—
9	PhotoTempControlP/0/stopDone	97	23.7
10	PhotoTempControlP/1/stopDone	0	
11	ArbiterP/3/granted	97	28.0
12	AdcP/acquiredData	97	42.3
Average			42.0

プさせ，タスクキューに実行待ちのタスクがある場合は（4）で実際に実行する．(3）と（5）が計測用に用意した関数であり，それぞれ実行開始時と実行終了時にタスク ID を汎用 I/O に出力する．タスクが `post` される時刻および割り込み処理に関しても同様の方法で計測を行った．

評価を行ったサンプルプログラムの 1 つである Sense[16] のタスク一覧と，各タスクの処理にかかる時間（Cost）を表 2 に示す．Sense は定期的にセンサ値をサンプリングし，下位 3bit を LED に表示するアプリケーションである．表 2 のタスク ID は TinyOS でタスクに付与される ID，Number of Times は測定期間内に当該タスクが実行された回数，Cost は当該タスクの実行に要した時間の平均である．最下段の Average は実行されたすべてのタスクの回数によって重みを付けた実行時間の重み付け平均である．Sense では，`AlarmToTimerC/0/fired` の 1 回の実行時間が 59.1 μs と最も大きく，全てのタスクの実行時間の平均が 42.0 μs となる．

同様の計測を Oscilloscope[17]，Blink[18]，RadioSenseToLeds[19]，MultihopOscilloscope[20]，RadioCountToLeds[21]，LowPowerSensing[22] のアプリケーションについて行い，タスク処理の平均コストを算出した．これらのアプリケーションは TinyOS のソースコードツリーに含まれているサンプルアプリケーションであり，実用的なセンサネットワークを動作させるうえで必要不可欠なタスクが含まれている．さらに，アプリケーション実行時に CPU が動作している時間とタスクの処理に要した時間から，全タスクに対するスケジューラのオーバヘッドの割合を算出した．

異なるアプリケーションにおけるタスク処理の平均コストとスケジューラのオーバヘッドを図 3 に示す．図 3 より，タスク処理のコストの平均が小さくなると，スケジューラ処理のオーバーヘッドの影響が大きくなることが分かる．例えば図 3 では Sense のタスクの平均コストが 41.92 μs と比較したアプリケーションの中で最も小さいものの，スケジューラのオーバヘッドは 19.77 %と最も大きくなる．タスクの粒度を小さくすればするほどスケジューラのオーバヘッドが増大することが示された．

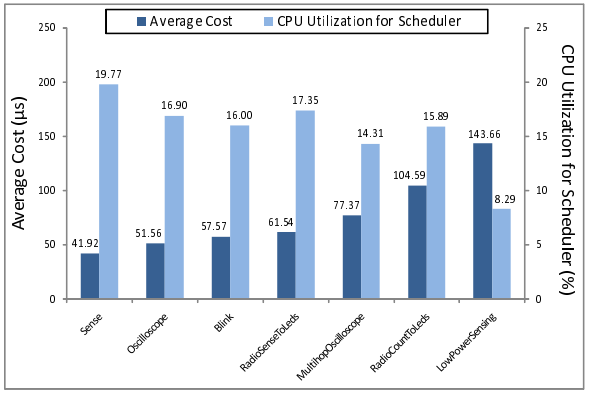

図 3 アプリケーション間のスケジューラのオーバヘッドとタスクの平均コスト

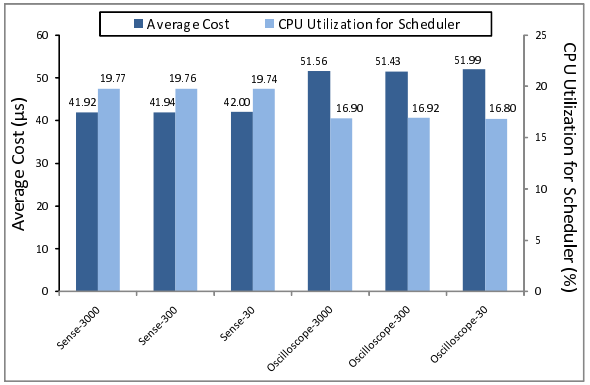

図 4 サンプリング間隔を変えた際のアプリケーション間のスケジューラのオーバヘッドとタスクの平均コスト

同一アプリケーションでタスク実行の周期を変化させた際の比較を図 4 に示す．横軸のアプリケーション名の末尾の数字はサンプリングタスクの実行周期を ms 単位で表してい

る．図 4 から，サンプリングの周期を変化させてもスケジューラのオーバヘッドの割合が変化しないことから，スケジューリングのオーバーヘッドは CPU の使用率ではなく，アプリケーションが含むタスクによって決まることがわかる．例えば Sense においてサンプリング間隔を 3000 ms，300 ms，30 ms と変化させてもスケジューラのオーバヘッドの割合は約 19.7 %と変化しない．

3.2 タスクの実行遅延

次に，計算量の大きなタスクを含めた際に発生する遅延の測定評価について示す．アプリケーションとして LowPowerSensing[22] を用い，無線モジュールの電源管理のタスクの遅延時間の測定を行った．LowPowerSensing では，定期的にセンサ値を取得し，取得したセンサ値をフラッシュメモリに保存する．また，LowPowerSensing はロングプリアンブル型の MAC プロトコル[11] を用いており，LPL（Low Power Listening）によりシンクノードからのセンサ値送信のリクエストを検知する．センサノードはシンクノードからのリクエストを受け取ると，フラッシュメモリに蓄積したセンサ値をシンクノードに送信する．

LowPowerListening の LPL 実行時の CPU と無線モジュールの動作を図 5 に示す．RF が ON となった後，シンクノードからの要求検知を行う `getCca` のタスクが起動する．`getCca` では，シンクノードからの送信要求が検出されない場合は終了時に RF OFF のタスクを `post` する．`getCca` の実行中にタイマイベントが発生し，ソフトウェアタイマのタスクが `post` されると，`getCca` 終了後にソフトウェアタイマ内に `event` として実装されたサンプリングのタスクが実行される．この時，タイマイベントである `event` の終了後に RF OFF のタスクが実行されるため，実行遅延が生じる．

ここで，サンプリングのタスク内に，0 ms，2 ms，4 ms，8 ms の大きさの計算処理を挿入し，タスクの実行開始の遅延を測定した．実験ではタスクの計算時間がタスク実行の遅延に与える影響を明らかにするため，サンプリングの周期と LPL の周期を遅延の発生しやすい 50 ms，13 ms に固定した．実行遅延の最大値と平均値を図 6 に示す．図 6 によれば，最大遅延は計算処理に費やす時間よりも多く発生している．例えば，タスクの処理時間が 8 ms の場合，最大遅延は約 8.78 ms となる．この遅延の増加は，タスクが長い場合にはタスクの実行中に他の割り込みが入る確率が増加することに起因する．このようなタスクの実行遅延の不確定性により，RF OFF の実行に予測不能な遅延が生じる．平均値に関しても，計算処理が大きくなるに伴い増加しているが，計算処理が 2 ms の場合と 4 ms の場合で平均値の大きさが逆転している．これは RF OFF のタスクの実行遅延により LPL の周期に変化が生じていることに起因する．

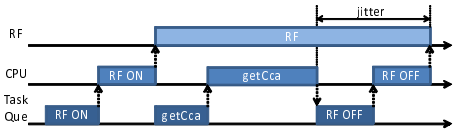

図 5 RF モジュールの電源管理

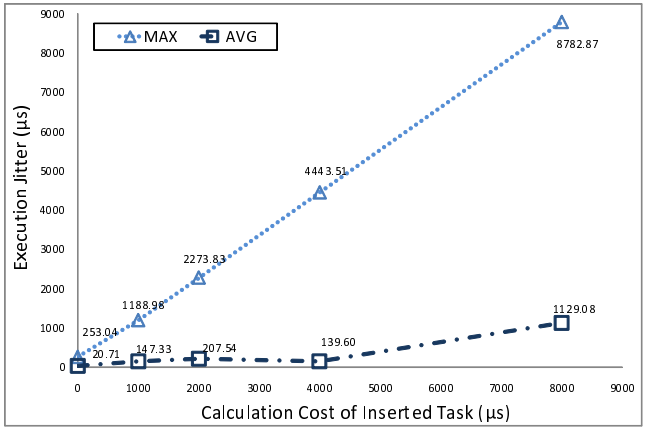

図 6 計算量を変化させた時のタスク実行遅延

このように，計算量の大きいタスクが存在すると，他のタスクの実行タイミングに不確定性が発生してしまう．例えば，無線通信モジュールの電力制御タスクの場合には，電源オフのタイミングが遅延することによる消費電力の増大や，電源オンのタイミングが遅延することによる予期せぬパケットロスなどが発生する．図 6 に示した 2 ms の場合と 4 ms の場合の平均値の大きさが逆転していることからも分かるとおり，タスクの遅延はあらかじめ予想できるものではなく，アプリケーションの構築において大きな問題となる．

4. タスクフローの解析

3. で述べた問題点に対して，筆者らは，CPU の消費電力を削減すると同時にリアルタイムタスクの実装を容易とする無線センサノード向けマルチコア CPU である Mulco の研究開発に取り組んでいる[6]．Mulco では，センサノードで実行されるタスクの実行要求が固定的かつ周期的であることに着目し，1 つのタスクフローを 1 つのコアに固定的に割り当て

る．このように，各タスクを複数のコアに分割することによって，タスク同士がコア内で影響を与え合うことを防ぐことができる．すなわち，1 つの CPU 資源を複数のタスクに配分する必要がないためのスケジューラが不要となる．さらに，割り当てられたタスクの時間制約を満たす最低の駆動周波数で各コアを動作させることができるため，低消費電力での動作が可能となる[6]．

Mulco では，ある特定の割り込み要求によって発生する一連のタスクごとにコアを用意する．本稿ではこの一連のタスクをタスクフローと呼ぶ．タスクフローごとにコアを用意するアプローチでは，アプリケーションを構成するタスクフロー数によっては必要となるコア数が膨大となる．コア数が増えすぎた場合，回路規模の肥大化や，コア間通信のオーバヘッドの問題が顕在化する．本節では，センサノードで実行されるタスクの実行フローを解析し，どの程度タスクを分散して各コアに割り当てられるのかの検証を行う．

タスクフローの初期的な検証として，TinyOS のサンプルアプリケーションである MultihopOscilliscope[20] のタスクフローの解析を行った．MultihopOscilliscope は定期的にサンプリングを行い，センサデータをマルチホップでシンクノードにリアルタイム転送するアプリケーションであり，サンプリング，無線送受信，ルーティング等，センサネットワークのアプリケーションに共通して必要とされるタスクの多くが含まれている．

表 3 に MultihopOscilliscope に含まれるタスクと測定時のタスクの実行回数，表 4 に割り込み要因および割り込み処理の実行回数を示す．表 3 と表 4 の各実行回数より，各処理がどのタスクフローに所属しているかが分かる．例えば，表 3 における実行回数が 381 回となっているタスク `ArbiterP/0/grantedTask` とタスク `PowerManagerP/0/startTask` は同じタスクフローに属する．また，表 3 における実行回数が 844 回であるタスク `AlarmToTimerC/0/fired` と表 4 における実行回数が 844 回である割り込み `TIMER0 Compare Match` は同じタスクフローとなる．

表 3 および表 4 の実行回数と，ソースコード内でのタスクの依存関係から解析したタスクフローを図 7 に示す．図 7 は `event` および `task` 間の要求関係と，次のタイマの発火設定など割り込みを発生させる `task`，`event` の因果関係から決定される．図 7 の四角は割り込みハンドラの処理，二重円はソフトウェアタイマなどの割り込みを受け付けシグナルを送信すべき適切なタスクフローを選択する処理，楕円は `task` と `event` での処理を表している．また，上から下への矢印では，`task` が `post` されており，横方向の矢印ではハードウェア割り込みは `event` として処理が遷移することを示している．

図 7(a) を例にタスクフローの説明を行う．このタスクフローでは 8 つのタスクと 2 つの

表 3 MultihopOscilloscope のタスク

TaskID	Task	Number of Times
1	AlarmToTimerC/0/fired	844
2	VirtualizeTimerC/0/updateFromTimer	1298
3	ArbiterP/0/grantedTask	381
4	ArbiterP/1/grantedTask	381
6	PowerManagerP/0/startTask	381
7	PowerManagerP/0/stopTask	381
10	PhotoTempControlP/0/stopDone	380
12	ArbiterP/3/grantedTask	381
13	AdcP/acquiredData	381
16	CC2420CsmaP/sendDonetask	78
23	CC2420ReceiveP/receiveDonetask	78
24	CtpForwardingEngineP/0/sendTask	153
27	CtpRoutingEngineP/0/updateRouteTask	6
28	CtpRoutingEngineP/0/sendBeaconTask	1
29	SerialP/RunTx	307
33	SerialDispatcherP/0/signalSendDone	153
37	UARTDebugSenderP/sendTask	154

表 4 MultihopOcsilloscope の割り込み

InterruptID	Interrupt	Number of Times
7	External Interrupt 6	231
11	TIMER2 Overflow	535
12	TIMER1 Compare Match A	159
14	TIMER1 Overflow	16
15	TIMER0 Compare Match	844
17	SPI Serial Transfer Complete	1081
20	USART0 TX Complete	3382
21	ADC Conversion Complete	381
29	TIMER3 Overflow	522

割り込み要求発生の待機から構成される．サンプリング周期ごとにタイマが発火し，センサの電源を ON にしてセンサの電源が安定するまでのタイマを設定する．設定したタイマが発火した後，AD コンバータに対して読み取り要求を発行し，変換完了割り込みが発生するまで待機する．AD 変換が完了したら割り込みが発生し，センサの電源をオフとしてタスクフローを終了する．

MultihopOscilloscope では，サンプリングに関連するタスクフロー（図 7(a)），ルーティングに関するタスクフロー（図 7(b)）および無線通信のデータ受信に関連するタスクフロー（図 7(c)）の 3 つの非同期なタスクフローが存在する．それぞれのタスクフローは独立した関係にあるため，このタスクフローごとにコアを割り当てると，コア内で非同期なタスクの実行要求が発生しない．

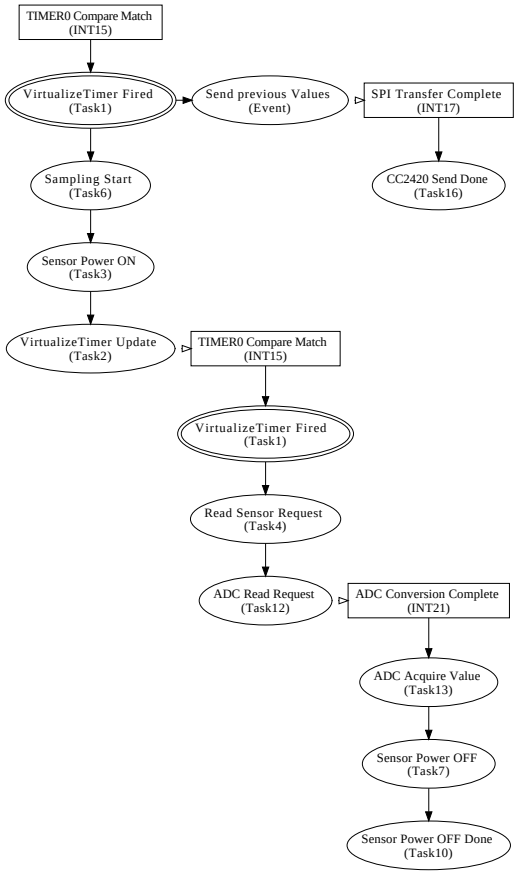

図 7-(a) サンプリングとデータ送信

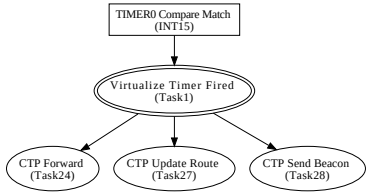

図 7-(b) ルーティング

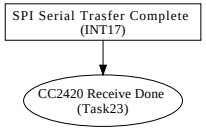

図 7-(c) データ受信

図 7 MultihopOscilloscope のタスクフロー

このタスクフロー図から，MultihopOscilloscope を Mulco で実装する際には，3 つのコアに割り振れば非同期な実行要求は発生せずに，実行遅延とスケジューラのオーバーヘッドの 2 つの問題を解決できると言える．ただし，図 7(a) においてはサンプリングのタスクフローとセンサ値送信のタスクフローが同期して起動する．2 つのフローは同期しているため同時に処理要求が生じることはないが，並列に実行される可能性があるため別のコアを割り当てることで消費電力を削減できる可能性が高い．無線センサノードでは 1 コアのコストが低く，特に現在のセンサノードでは CPU ダイの面積はほぼ SRAM によって占められているため，数個のコア数であればノードのコスト増加は低く抑えられる．

5. 考　　察

5.1 消 費 電 力

CPU のマルチコア化によりスケジューラを除去することで，ノード全体の消費電力の 10 %程度を削減できる．4. のタスクフローの分析において，CPU のマルチコア化によりタスクスケジューラを排除できることを示した．スケジューリングより CPU で消費される電力はアプリケーションごとに異なるが，タスクの計算量が小さいアプリケーションにおいては約 20 %の電力をスケジューラ処理で消費していた．センサノード全体の消費電力のうち，50 %を CPU で消費している[8] ことを考えると，ノード全体の約 10 %の電力をスケジューラが消費していることになる．

Mulco では駆動周波数を低減させ，電源電圧を低く設定することでさらに消費電力を削減することができる．3. のタスクの遅延の評価では，タスクの実行遅延の最大値はタスク処理にかかる最大の時間に応じて増加した．現在のシングルコアの CPU システムでは発生しうる遅延の最大値を考慮して CPU の駆動周波数を選択しなければならない．本稿の評価では，タスクの実行遅延の傾向を把握するため，意図的に遅延の発生しやすいタスクの実行周期を選択したが，タスクの実行周期を変更しても発生しうる遅延の最大値は変化せず，タスク処理にかかる最大の時間により決定される．これに対して Mulco では，CPU のマルチコア化によりスケジューリングにより生じていたタスク実行の遅延を排除できる．また，Mulco では同一コア内で同時に 2 つの処理要求が生じないため，先発タスク存在によるタスクの実行遅延が発生しない．

5.2 ハードウェアサポート

CPU にハードウェアサポートを導入することで，スケジューリングのオーバーヘッドを大きく削減することができる．例えば，Ekanayake らが行った無線センサノード向け非同

期 CPU の研究[12] では，スケジューリングのオーバーヘッドが無視できないことを指摘し，TinyOS と同じく FIFO のタスクスケジューラをハードウェアで実現することで CPU をスケジューリングから解放している．

Mulco では，コア内でのスケジューリングは不要となるが，コア間でのスケジューリングは依然として必要である．例えば，3.2 で示した通り，ソフトウェアタイマを利用する場合には，発生した割り込みがどのタスクフローに対応するものなのかを判断する必要がある．この条件判断は柔軟に記述できることが好ましい．Protothreads[13] ではイベントモデルの TinyOS に疑似的にスレッドを導入し，条件ブロックを可能とすることでソフトウェアの記述性を大きく向上させている．Mulco ではこの条件判断を Protothreads のようにソフトウェアで行うことも可能ではあるが，効率的かつ柔軟に行うことができるハードウェアを設計することができれば，さらに効率化することができる．

6. おわりに

本稿では，無線センサノードにおけるシングルコア CPU の問題点の定量的な評価を示した．また，アプリケーションのタスクフローの解析によって，Mulco が現実的な解決策になりうることを示した．現在，今回の評価に基づき，Mulco の詳細な設計に取り組んでいる．また，5. でも述べたように，ハードウェアサポートの要件抽出および設計も必要である．他のアプリケーションのタスクフロー解析を行うとともに，排他制御機構やデータ共有機構などの検討を行っている．

参考文献

1) 鈴木 誠, 猿渡 俊介, 南 正輝, 森川 博之, “無線センサネットワークにおける時刻同期技術の研究動向”, 森川研究室 技術研究報告書, No. 2008001, 2008.
2) 森戸 貴, 猿渡 俊介, 南 正輝, 森川 博之, “無線センサネットワークを用いたアプリケーションに関する研究動向”, 森川研究室 技術研究報告書, No. 2008002, 2008.
3) P.Levis, D.Gay, V.Handziski, J.Hauer, B.Greenstein, M.Turon, J.Hui, K.Klues, C.Sharp, R. Szewczyk, J. Polastre, P. Buonadonna, L. Nachman, G. Tolle, D. Culler, and A. Wolisz, “T2: A Second Generation OS For Embedded Sensor Networks”, In *Technical Report TKN-05-007, Telecommunication Networks Group, Technische Universitat Berlin*, 2005.
4) *MTS300 Data Sheet*: `http://www.xbow.com/Support/Support_pdf_files/MTS-MDA_Series_Users_Manual.pdf`
5) *MicaZ Data Sheet*: `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`
6) S.Ohara, M.Suzuki, S.Saruwatari, and H.Morikawa, “A Prototype of a Multi-core Wireless Sensor Node for Reducing Power Consumption”, In *Proceedings of the 2008 International Symposium on Applications and the Internet (SAINT'08)*, Turku, Finland, 2008.
7) C.Park, Q.Xie, P.Chou, M.Shinozuka, “DuraNode: wireless networked sensor for structural health monitoring”, In *Proceedings of the 4th IEEE International Conference on Sensors*, Irvine, California, USA, 2005.
8) V. Shnayder, M. Hempstead, B. Chen, GW. Allen, M. Welsh, “Simulating the power consumption of large-scale sensor network applications”, In *Proceedings of the 2nd ACM Conference on Embedded Network Sensor Systems (SenSys'04)*, Baltimore, Maryland, USA, 2004.
9) S.Kim, S.Pakzad, D.Culler, J.Demmel, G.Fenves, S.Glaser, and M.Turon, “Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks”, In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, Cambridge, Massachusetts, USA, 2007.
10) S. Saruwatari, M. Suzuki, and H. Morikawa, “A Compact Hard Real-Time Operating System for Wireless Sensor Nodes”, In *Proceedings of 6th International Conference on Networked Sensing Systems (INSS 2009)*, Pittsburgh, Pennsylvania, USA, 2009.
11) J.Polastre, J.Hill, and D.Culler, “Versatile Low Power Media Access for Wireless Sensor Networks”, In *Proceedings of the 2nd ACM Conference on Embedded Network Sensor Systems (SenSys'04)*, Baltimore, Maryland, USA, 2004.
12) V. Ekanayake, C. Kelly IV, R. Manohar, “An ultra low-power processor for sensor networks”, *ACM SIGARCH Computer Architecture News, Volume 32, Issue 5*, 2004.
13) A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, “Protothreads: Simplifying event-driven programming of memory-constrained embedded systems”, *Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys'06)*, Boulder, Colorado, USA, 2006.
14) `http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x/`.
15) `http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x/tos/system/SchedulerBasicP.nc/`.
16) `http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x/apps/Sense/`.
17) `http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x/apps/Oscilloscope/`.
18) `http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x/apps/Blink/`.
19) `http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x/apps/RadioSenseToLeds/`.
20) `http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x/apps/MultihopOscilloscope/`.
21) `http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x/apps/RadioCountToLeds/`.
22) `http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x/apps/tutorials/LowPowerSensing/`.