Power Analysis Guide for Windows*

Revision 1.0

Authors: Khang Nguyen, Vikram Shankar

Contributors: Rajshree Chabukswar, Manuj Sabharwal, Belinda Liviero
# Table of Contents

1. **INTRODUCTION** .......................................................................................................................... 3  
   1.1 PURPOSE .................................................................................................................................. 3  
   1.2 ANALYSIS CYCLE ....................................................................................................................... 3  

2. **LOGGING AND PROFILING** ..................................................................................................... 4  

3. **DOES MY APPLICATION HAVE AN ISSUE?** ............................................................................. 5  
   3.1 PERCENTAGE C-STATE RESIDENCIES ...................................................................................... 5  
   3.2 PLATFORM TIMER RESOLUTION ............................................................................................... 6  
   3.3 INTERRUPT RATE ...................................................................................................................... 7  
   3.4 CONTEXT SWITCHING RATE ...................................................................................................... 7  
   3.5 SYSTEM CALL RATE ................................................................................................................... 8  

4. **IDENTIFYING ROOT CAUSE** .................................................................................................. 8  
   4.1 PLATFORM TIMER RESOLUTION ............................................................................................... 8  
   4.2 PERIODIC ACTIVITIES IN IDLE MODE ....................................................................................... 9  
   4.3 USE OF SLEEP API .................................................................................................................... 10  
   4.4 USE OF WAITFOROBJECTS API .............................................................................................. 11  
   4.5 USE OF PEEKMESSAGE API ...................................................................................................... 12  
   4.6 POLLING .................................................................................................................................... 12  
   4.7 SPIN WAIT WITHOUT PAUSE ................................................................................................... 12  
   4.8 NON-CONCURRENT CPU AND GPU ACTIVITIES .................................................................... 13  
   4.9 LACK OF TIMER COALESING .................................................................................................... 13  
   4.10 EXCESSIVE I/O ACTIVITY ......................................................................................................... 14  

REFERENCES ..................................................................................................................................... 14
1. Introduction

1.1 Purpose
Finding where an application consumes power can be very challenging. The purpose of this guide is to provide easy-to-follow step-by-step guidance to developers on how to find and troubleshoot power-efficiency issues in their applications. We recommend that the developer read the guide in its entirety at least once to get a feel for how to go about finding and fixing power related issues. You may then use the guide as a quick reference companion that you can refer to while you go about troubleshooting your application.

The scope is limited to Windows 7* and Windows 8*. We have tried to cover as many of the common power related issues as possible. We will continue to add to and refine the manual. Your suggestions are certainly welcome!

Of note here is that the guide does not cover how to go about measuring power consumption. Though power consumption information is certainly useful for detailed analysis, it is not necessary for finding and fixing the most common power related issues in an application.

1.2 Analysis Cycle
There are several distinct phases involved in profiling and troubleshooting an application for power issues. Ideally, developers must consider power-efficiency an integral requirement for their applications and make every effort to choose power-efficient algorithms and language constructs from the very start. More information on how to optimize an application for power-efficiency can be found at [1]. Once an application has been written, we must profile it to see if there were any power-issues that crept in during development. Section 2 (Logging and Profiling) introduces the tools and methodology we will use to collect data about an application. Section 3 (Does my Application have an Issue) discusses a few metrics that can be used to quickly determine if the application may have an issue. Section 4 (Identifying Root Cause) provides step-by-step guidelines on how to identify the root cause of an issues and suggestions on how to fix it.
2. Logging and Profiling

We begin by collecting and logging data about the application being profiled. Data we are interested in include processor C-state residencies (more on this later), interrupt rates, context switch rates, call stacks etc. There are several tools you can use to collect this data but we will use these three:

1. **Performance Monitor (PerfMon):** is a Windows tool used to view performance data. Although the name of the tool implies that it is only for performance, it also provides useful information that can be used for power analysis: CPU utilization (% processor time), Interrupt Rate, Context Switching rate, and System Call Rate. In Windows 8, it also provides information like clock interrupt rate and the wake up rate. You can find more information on PerfMon at [4].

2. **Battery Life Analyzer (BLA):** is a software tool that monitors various software and hardware activities that affect battery life. We will use BLA to quickly determine if our application even has a power-efficiency issue that needs further analysis. Please refer to [1] for more information on how to use BLA.

3. **Microsoft Windows Performance Toolkit (Xperf):** consists of the Windows Performance Recorder (WPR) to collect data, and the Windows Performance Analyzer (WPA) to analyze the data. WPR is based on Event Tracing for Windows (ETW) and captures detailed system and application behavior, and resource usage. WPA on the other hand is used to review the event trace log files created by WPR. It displays the performance data in graphs and tables, making it easy to investigate potential issues. With this toolkit you can root cause an issue by drilling down to the process, thread, and API level to find the power hungry calls in the application. An introductory tutorial on how to use this toolkit can be found at [3].

Profiling and analysis must always be done on a “clean” system. A clean system here means that it only runs applications or services that are needed to operate the system and to run the target application. Applications like antivirus software and disk utilities should be disabled or removed.

An application must be profiled under two scenarios:

- **Idle Mode:** An application is in the idle mode when it is running without doing anything. For example, a media player is in the idle mode when it is running but is not playing a video clip. It is very important to do this analysis because many user applications spend much of their time in this mode. An idle application should only consume minimal resources.

- **Active Mode:** The application is in the active mode when it is running a workload. For example, a media player is in the active mode when it is running a video clip. In this mode we are generally interested in analyzing performance issues impacting power as well as wasted work done impacting overall energy consumption on the system.

**Data logging steps are as follows:**
1. Start PerfMon. The counters of interest are: Deepest C-State, Interrupt Rate, Clock Interrupt Rate, Wake-up Rate, Context Switch Rate, and System Call Rate.
2. Start your application and wait for fifteen minutes for the system to settle down before collecting data.
3. Collect PerfMon data for at least 3 minutes. Save the results and close PerfMon.
4. Start BLA. Set it up to collect data for the following modules: “Software Activity Analysis” and “CPU C-States”
5. Wait for the system to settle before logging. Collect data for at least 3 minutes.
6. Review PerfMon and BLA data to see if your application needs further analysis. Please refer to section “Does my Application have an Issue?” for details. If no issues, you are done.
7. Close BLA and start WPR.
8. Wait for fifteen minutes for the system to settle.
10. Refer to section “Identifying Root Cause” to find what may be causing issues in your application.

3. Does my Application have an Issue?

Once we have collected data on our applications we must be able to quickly decide if we even have a power efficiency problem. This section covers some common metrics that we can use to decide whether our application is power-efficient or not.

3.1 Percentage C-State Residencies

Modern processors implement various power saving states called C-states. A processor is in C0 state when it is active. The Operating System can put individual cores in the processor into deeper C-states as needed. As a core enters deeper C-states (for example, from C0 to C3 to C7) it becomes more power-efficient by turning off more and more functions. The flip side is that a core takes longer to exit deeper C-states and hence there is a latency penalty. The “Package C-state” is the shallowest C-state amongst the cores in a multi-core processor. In general, an idle application should allow the processor to enter its deepest C-state and stay there as long as possible. Hence “C-State Residencies” – the amount of time (as a percentage) the processor spends in various c-states – are a very important metric of application power-efficiency. It is also valuable to observe the frequency with which a processor enters/exits various c-states; generally, less power will be consumed when the frequency of transitions is reduced. More information about C states can be found at [4].

It is recommended that you use BLA to collect C-state residencies. PerfMon and Xperf report up to C3 while BLA can report even C7 residencies. Further, BLA reports the actual C-state residencies.
(called “hardware C-states”) while the other tools report only estimation (called “software C-states”).

**Recommended Values**

<table>
<thead>
<tr>
<th></th>
<th>For Idle Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Package C0-C1</td>
<td>&lt; 5%</td>
</tr>
<tr>
<td>• Package C2</td>
<td>&lt; 2%</td>
</tr>
<tr>
<td>• Package C7</td>
<td>&gt; 95%</td>
</tr>
</tbody>
</table>

There are no recommended values for an active application. It really depends on the category of application — if it is really heavily CPU-bound, such as an unlocked encoder, then it may show 100% C0 residency.

**Potential Causes**

<table>
<thead>
<tr>
<th></th>
<th>How to Detect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. High Platform Timer resolution</td>
<td>See Section 4.1</td>
</tr>
<tr>
<td>2. The use of PeekMessage without WaitMessage to process Windows messages</td>
<td>See Section 4.5</td>
</tr>
<tr>
<td>3. Non-concurrent CPU and GPU Activities</td>
<td>See Section 4.8</td>
</tr>
<tr>
<td>4. Lack of timer coalescing in multithreaded applications</td>
<td>See Section 4.9</td>
</tr>
<tr>
<td>5. Excessive I/O</td>
<td>See Section 4.10</td>
</tr>
</tbody>
</table>

### 3.2 Platform Timer Resolution

The operating system uses the Platform Timer to wake up the CPU and schedule processes. On Windows 7 this timer is usually triggered every 15.6ms while on Windows 8 the Operating System dynamically sets the timer resolution based on scheduling needs. Both Operating Systems allow applications to change the platform timer resolution so that they can better handle any time-critical tasks. For example, a video conferencing application may change the timer resolution to 1ms so that it can receive and decode frames as soon as possible. In general, a higher platform timer resolution will consume more power because the CPU is woken up more frequently. Use BLA to look at the platform timer resolution and to see if your application is changing it while idle and/or active.

**Recommended Values**

- An idle application rarely needs a timer value of less than 15.6ms. If your application changes the timer value during active processing, ensure that it restores it back to 15.6ms when it goes to idle.
### Potential Causes and How to Detect

<table>
<thead>
<tr>
<th>Potential Causes</th>
<th>How to Detect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. An application increased the timer resolution while in the active mode but did not restore it when going into idle</td>
<td>See Section 4.1</td>
</tr>
</tbody>
</table>

#### 3.3 Interrupt Rate

Interrupt Rate is the average rate, in incidents per second, at which the processor receives and services hardware interrupts. This value is an indirect indicator of the activity of devices that generate interrupts, such as the system clock, the mouse, disk drivers, data communication lines, network interface cards, and other peripheral devices. We get this value from PerfMon.

**Recommended Values**

- In general, the interrupt rate will need further investigation if it is greater than 8000-9000 interrupts/sec in active mode. Take a look at device activity (graphics, USB, network etc.)

<table>
<thead>
<tr>
<th>Potential Causes</th>
<th>How to Detect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Excessive I/O activity (Most likely cause)</td>
<td>See Section 4.10</td>
</tr>
<tr>
<td>2. Short timer resolution</td>
<td>See Section 4.1</td>
</tr>
<tr>
<td>3. Longer DPC call.</td>
<td>Use xperf</td>
</tr>
</tbody>
</table>

#### 3.4 Context Switching Rate

Context Switching Rate is the combined rate at which all processors on the computer are switched from one thread to another. Context switches occur when a running thread voluntarily relinquishes the processor, is preempted by a higher priority ready thread, or switches between user-mode and privileged (kernel) mode to use an Executive or subsystem service. A high rate of context switching means that the processor is being shared repeatedly. High context switching causes more overhead and prevents the system from going idle thus increasing power consumption. Try to reduce this value by decreasing the number of active threads. We get this value from PerfMon.

**Recommended Values**

- Do further investigation if the Context Switch Rate is greater than 7K per active thread in the application.
1. Platform timer resolution is high  
   See Section 4.1

2. Busy Wait  
   See Section 4.7

3. Use of sleep API with low timeout value (e.g. sleep(0)), 
   WaitForSingleObject with short timer period  
   See Section 4.3

3.5 System Call Rate
System Call Rate is the combined rate of calls to operating system service routines by all processes running on the computer. These routines perform all of the basic scheduling and synchronization of activities on the computer, and provide access to non-graphic devices, memory management, and name space management. Again, in general, if this value in the active mode is three times greater or equal to that in the idle mode then it is unusual high. We get this value from PerfMon.

Recommended Values
- Do further investigation if System Call Rate is greater than 45K per active thread in the application. See reference [10] for more details.

<table>
<thead>
<tr>
<th>Potential Causes</th>
<th>How to Detect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Platform timer resolution is high</td>
<td>See Section 4.1</td>
</tr>
<tr>
<td>2. Use of WaitForSingleObject API with short timer period</td>
<td>See Section 4.4</td>
</tr>
<tr>
<td>3. Excessive I/O activity</td>
<td>See Section 4.10</td>
</tr>
<tr>
<td>4. Frequent calls to device drivers</td>
<td></td>
</tr>
</tbody>
</table>

4. Identifying Root Cause
In this section we will show how to identify the root cause of an issue. For each issue we will also make some recommendations on how to fix it.

4.1 Platform Timer Resolution
Some applications purposely change the platform timer resolution to 1ms to ensure smoothness of operation. This will cause the system to wake up more often, thus consuming more power. BLA can be used to identify the process that changes the platform timer resolution.

Detection Method:
1. In BLA select and run the module “Software Activity Analysis” if you haven’t already done so.
2. Select the “Active Analysis” tab to display the timer tick. In the example below we highlight one sample application where the platform timer period was changed to something other than the default.

![Battery Life Analyzer 2.3](image)

**Problem Resolution**

1. Do not change the platform timer resolution unless it is absolutely necessary. If changing the timer is unavoidable, make sure to change it back to 15.6ms when no longer required.
2. If the application needs a high resolution time stamp, use `QueryPerformanceCounter` or `RDTSC` API instead of `GetXXXTime` APIs.

### 4.2 Periodic Activities in Idle Mode

Similar to the above problem, periodic activities with short intervals will cause the system to wake up more often. WPA can be used to identify periodic activities.

**Detection Method:**

1. Using WPA, open the data (ETL file) you logged using WPR in section 3. Select the option “Computation”
2. Then select “CPU Usage (precise) By Process, Thread”
3. Select option “Timeline by Process, Thread”.
4. See if the chart shows what appears to be periodic activity. For example, in the figure below we see a periodic activity at approximately every 11ms.
5. Look at the data table to see which specific thread is causing the periodic activity. For example, in the figure below, column TimeSinceLast shows that thread 3644 is activated every 10,905 microseconds.

Problem Resolution
- Minimize the use of periodic activities. If periodic activities are indeed needed, use the largest time period your application can tolerate.
- Reduce frame refreshing or render activities in Idle Mode since it can impact the display power.

4.3 Use of Sleep API
The Sleep API is used to suspend the current thread for a period of time. The Sleep API call with zero duration is commonly used to context switch the calling thread out if another thread of the same or higher priority is ready. However, if this statement is called very often, it will increase the context switch rate and cause the power consumption to increase.

Detection Method:
Use WPA to detect calls to sleep().

Problem Resolution
- If Sleep(0) is being used as a method synchronize multiple threads in an application consider the use of mutex/semaphores instead.
• If `Sleep(0)` is being used as a method to balance load amongst worker threads consider using it in conjunction with `pause` API. More details can be found in reference [8].

4.4 Use of `WaitForSingleObject` API

The `WaitForSingleObject` API, as its name implies, is to wait until the specified object is in the signaled state or the time-out interval elapses. If this API is called with a very short time interval, it will impact the battery life. WPA can detect where this API is called in an application.

Detection Method:
1. In WPA, select the option “Computation”
2. Then select “CPU Usage (precise) By Process, Thread”
3. Select option “Timeline by Process, Thread”.
4. See if the chart shows what appears to be periodic activity.
5. Double-click on the thread that causes periodic activities.
7. Look for calls to `WaitForSingleObject` in the call stack.

Problem Resolution
• See if you can use `TryEnterCriticalSection` with spin count instead (using it with no spin count may not be that efficient.) Please see reference [6] for details.
4.5 Use of PeekMessage API
This API is used to dispatch incoming sent messages, check the thread message queue for a posted message, and retrieve the messages if they exist. If there is no message waiting, it will immediately return. Using PeekMessage in a loop will keep the system in the active mode and will consume a lot of power.

Detection Method
Use WPA to look at the call stack and see if PeekMessage appears frequently in the area where the application should have been idle. (Follow steps similar to those in section 4.4).

Problem Resolution

4.6 Polling
Polling, generally, refers to the situation where a device is repeatedly checked for readiness, and if it is not, the computer returns to a different task. Polling is not good for power since it will keep the CPU active, thus consuming unnecessary power.

Detection Method
This depends on calls of polling APIs. Under WPA use call stacks to analyze to see if polling is done excessively.

Problem Resolution
Replace polling logic with event-driven logic whenever possible. For example, use event-driven to alert that data is arrived at certain I/O port instead of constantly checking that I/O port for data.

4.7 Spin Wait without PAUSE
Spin wait or busy-wait is very similar to polling. Unlike polling, spin-wait checks to see if a device is ready, and if it is not, the computer doesn’t return to a different task. Spin wait without using the pause API will waste power.

Detection Method
1. Busy wait often shows a high context switch rate relative to CPU utilization.
2. If debugging ring0 applications, you will see NTDelayExecution in the call stack.

Problem Resolution
Insert the pause statement into the spin-wait loop.
4.8 Non-concurrent CPU and GPU Activities

Detection Method
This is detected using GPUView. The figure below shows an example of non-concurrent CPU and GPU activities.

Problem Resolution
Consider using `SetMaximumFrameLatency()`

4.9 Lack of Timer Coalescing
This feature was introduced in Windows 7 and allows scheduled timers to specify a tolerable delay for timer expiration. This allows the OS to group multiple software timer expirations into a single period of processing. This way the CPU gets interrupted less often, thus saving power. More information about this feature can be in reference [9].

Detection Method
Under WPA use overtime views of threads. See if different threads are active at different times.

Problem Resolution
Allow timer coalescing with a time latency that the application can tolerate.
4.10 Excessive I/O Activity

Detection Method
High Network I/O, USB or storage activities are indicated by high package C2 residencies (as observed by BLA.) You can then use WPA to view which specific IO device is causing the issue.

The HDD Spin-down module in BLA can be used to see if the disks stay spun-up most of the time. If the HDD Spin-down analysis is showing that the HDD is spending a lot of time spun-up, then the Disk Activity modules’ analysis can be used to identify the software that is causing the disk spin-up.

Problem Resolution
Reduce the I/O activities by the application when feasible. Always read/write a large chunk of data instead of doing it many time with small set of data.

References
5. Alon Naveh, Efraim Rotem, Avi Mendelson, Simcha Gochman, Rajshree Chabukswar, Karthik Krishnan, Arun Kumar, “Power and thermal management in the Intel® Core™ Duo processor”,
available online


**About the Authors**

The authors are Application Engineers working within the Scale Engineering organization of the Developer Relations Division at Intel, each focusing on software enabling for the developer.
Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel, the Intel logo, VTune, Cilk and Xeon are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others

Copyright© 2012 Intel Corporation. All rights reserved.

Optimization Notice


Performance Notice

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks