# Power and Performance Analysis of PDA Architectures

Robert Lee
Ripal Nathuji

December 12, 2000

**Abstract**

As Personal Digital Assistant (PDA) usage increases, so does the demand for better performance while adhering to strict power consumption limitations. As of the time of this study, the majority of PDA usage involves user-level applications which are not yet as computationally intensive as their desktop PC workstation counterparts. Because of that, it seems inappropriate to apply common benchmarks to these architectures, as it would not yield a particularly relevant analysis of how well the performance of these machines meets with current needs. Accordingly, new benchmarks specifically targetting current usage patterns must be devised. By analyzing the performance patterns and power usage footprint of codes that specifically target common user tasks and hardware subsystems (such as LCD display, audio, etc) one can isolate areas of weakness in present architectures.

# Introduction

The growing importance and prevelance of mobile computing devices underscores the need for low-power computing. Increased dependence on the battery forces hardware designers to both use less power and be able to satisfy increasingly intensive performance requirements. To this end, PDA architechtures cannot be accurately judged by existing standards. The aim of this paper is to present and employ a standard methodology for evaluating PDA's that is more fitting to the specific constraints that they face.

# Methodology

## 1. Approach

Based on usage patterns we chose to isolate three areas to examine for performance. Because of the current niche that the portable devices that we are examining hold, it is imperative that they be capable of relatively fine-grained graphics. Consumers expect a high-level graphical user interface (GUI) similar to Microsoft Windows. This is evidenced by the fact that most PDAs are currently sold running the Microsoft Windows CE operating system. Because of that, we decided that the CE operating system was a good platform on which to conduct our benchmark measurements because it both reflected current usage, and would allow for a standard platform to be used across our samples.

The four PDAs that we chose to investigate are the Compaq Aero, the Compaq iPAQ, the Hewlett Packard Jornada, and the Casio Cassiopeia. The Aero is powered by the MIPS R4000 processor at a rate of 70 MHz and 16 MB of RAM. The iPAQ is powered by an Intel StrongARM processor at a rate of 206 MHz and 32 MB of RAM. The Cassiopeia is powered by a MIPS VR4121 processor at a rate of 131 MHz and 32 MB of RAM. Finally, the Jornada is an SH3 based PDA clocked at 133 MHz and has 32 MB of RAM.

To reflect the importance of supporting such a GUI, we decided to write a benchmark to measure the performance(speed) of running common windowing tasks. This includes repositioning windows, toolbars and other elements as well as repainting and refreshing the screen.

Another important usage pattern that we noted was the high rate of task switching. Because of the consumer applications that are being used on these machines, it is often necessary to switch from one application, to another. For example, in the course of one logical transaction, it may be necessary to enter the "tasks list" and then switch to the "calendar" and then perhaps to the "address book" all in the course of jotting down a quick business meeting. To this end, we wrote another benchmark program that selects from the list of active threads running in CE and cycles between them an arbitrarily large number of times. The aim of this is to measure the speed of the context switch.

Lastly, with the growing importance of supporting Java applications especially in embedded systems, we thought it would be appropriate to rate the performance of exisiting JVM's on our PDA samples. We chose a popular VM (and more importantly one that supported all of our architectures) called CrEme from NSIcom. By measuring the performance of the VM by using an exisiting benchmark suite, selected codes from SPECjvm98 trimmed down to meet memory limitations, we aim to evaluate how well each PDA is equipped to run the increasing number of java codes.

The other aspect of PDA's under investigation was power. This includes both high level power measurements during power on and other runtime modes, as well as lower level measurements to estimate the power consumption of specific machine code sequences. These measurements were taken directly from the DC power source to the PDA's. Our choice of methodology carried both advantages and disadvantages. Our initial goal was to be able to observe the changes in power consumption at the instruction level granularity. Unfortunately, due to the intrinsic nature of electronic devices to act as lowpass filters, we could not obtain precise changes in power consumption during runtime. On the otherhand, our approach provides an acurate picture of the power consumption of the PDA package as a whole. This is appropriate because although

the details of the processor architecture are important to the performance of the PDA, this study takes into consideration the ability of commercial PDAs in their enirety.

# Results

## 1. Performance

The performance benchmarks that we developed for use across the PDAs differ from the standard set of scientific application benchmarks. The motivation for deviating from the standard toolkit of benchmarks is an analysis of usage patterns [HN98]. By developing this set of benchmark codes with portablility and cross-platform compatibility in mind as well as some notions of consumer usage, we aimed to accurately critique the ability of current commercially available hardware to satiate current consumer usage patterns.

To that end, we introduce four performance metrics by which to consider these handheld platforms, two custom applications aimed to heavily exercise the ability of the platform to perform task switching and GUI windowing operations.

Given what we see as a trend in the computer industry today, we feel that increasingly, Sun's Java technology is playing an increasingly important role in computing, and specifically in embedded systems. In examining the plausibility of supporting Java codes on PDAs, we decided to use SPECjvm98 as a metric for comparison. Using a popular Virtual Machine (VM) implementation called CrEmE, we ran selected codes from the SPEC suite and compared the performance on these codes across the different machines. The codes that we selected were jess, jack and mpeg. Our fourth metric was decidedly less quantitative, but nonetheless important. We qualitatively judged the performance and quality of playback of a given MPEG video clip. The motivation in this was very much along the same lines as the other metrics in so far as to attempt to form an accurate heuristic by which to guage the various handhelds that we studied.

### 1.1 Task Switching

The aim of this code was to target the ability of of the handheld to be able to task between active current running threads seemlessly. This is an important operation that needs to be able to occur quickly and efficiently. This is reflected by the fact that the Windows CE Operating System (OS), that powers the handhelds, supports 256 thread priority levels, whereas the desktop/server counterpart Windows NT only supports 8.

The task switching code works by enumerating the top ten active window handles in the system queue and arbitrarily cycling through them activating each in turn. This loop is run an arbitrarily large number of times while the runtime is recorded.

### 1.2 GUI Window Operations

This code is targetted towards assessing the performance hit associated with maintaining and manipulating the heavy graphical user interface associated with the Windows Operating Systems. The need to support high-end elements such as scroll bars and movable windows is an inherent requirement for a good handheld platform, as consumers become more and more accustomed to the Windows 'look and feel'.

To that end, this benchmark code starts by creating a window containing multiple standard Windows controls (such as textfields, command buttons, etc...) The code then repositions the window and its elements to various screen positions an arbitrarily large number of times causing furious screen refreshing and repainting while the runtime of this sequence is recorded.

### 1.3 SPECjvm98 Benchmark Suite

The SPECjvm98 suite that we looked at had many interesting codes that test various abilities of VMs. However, on inspection, running many of the codes on the handhelds proved to be infeasible due to memory limitations. According to the SPEC documentation, many of the benchmarks utilize heaps of sizes greater than the total memory on the handhelds. For this reason, we selected three codes that had a low memory profile but yet were still interesting enough to merit measurement. Memory was also a consideration in that each of the benchmarks have input files of various sizes, many of which were too large to run.

**jess** The jess benchmark is meant to simulate an expert shell system. The input is a file containing a list of rules and assertions which the expert system can then convert and solve.

**jack** The jack benchmark is modelled after parser generators such as yacc, lex, java_cup and the like. The input file to jack is a specification of the grammar to be parsed.

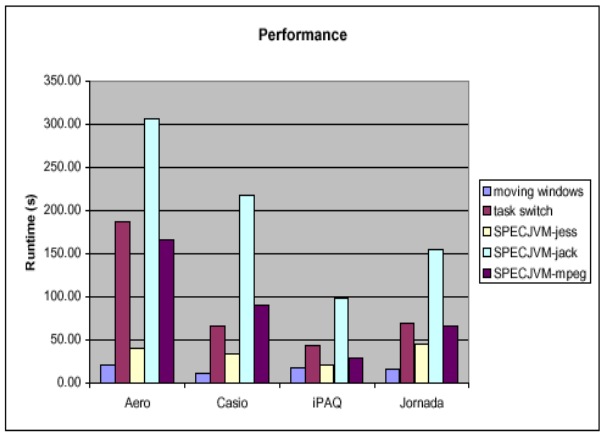
**mpeg** The mpeg benchmark decodes MPEG layer-3 audio samples

### 1.4 MPEG Video playback

The final performance metric that we propose is MPEG video playback. This kind of functionality will become increasingly important, as mobile computing becomes more and more widespread. For this metric, we subjectively judged the quality of playback, on a 1 MB video file.

The original goals for this metric included an investigation of the performance of playing back streaming audio and video. However, due to logistical concerns, we weren't able to investigate this thoroughly, though this remains an area of interest.

### 1.5 Results



In drawing conclusions from our performance testing, we felt it necessary to keep comparisons of the GUI testing and the JVM testing separate. The reason behind this is that though each set of tests are designed to provide a means of comparison, there is no direct correlation between the two that could let us make any conslusions.

In studying the results of the GUI operations, it can be seen that all the different platforms performed better on simple window operations than on task switching. In particular, the Aero did extremely poorly with a runtime nearly nine times greater in the task switching benchmark than in the window operations benchmark. In hypothesizing a reason for these results, we find it important to first realize a key difference between PDA architectures and the architectures found in most desktop computers. These PDA systems have a proportionally much smaller data cache to memory ratio (8KB compared to something more on the order of 1MB on desktop machines). The reason that this could decrease performance is that when running multiple processes, there is proportionally less cache space for each process in PDA's than on desktops. We feel that this could very well be the reason for our results.

The JVM results also lead us to come to a few conclusions. Overall, all of the platforms perform the jack behchmark extremely poorly. We believe strongly that the poor performance on jack was due to the necessity of accessing large input files. During our initial stages of testing, we ran the SPECjvm benchmarks in their full capacity. Under this condition, jack ran for several hours on the Aero without reaching completion. Through debugging code, we observed that the large percentage of delay occurred when the benchmark had to load an input file. Because of this we ran the benchmarks at a scaled down level so that they could successfully run on the PDA's. The correlation of loading large files and reduced performance leads us to question the efficiency of memory accessing on these systems.

## 2 . Power Consumption

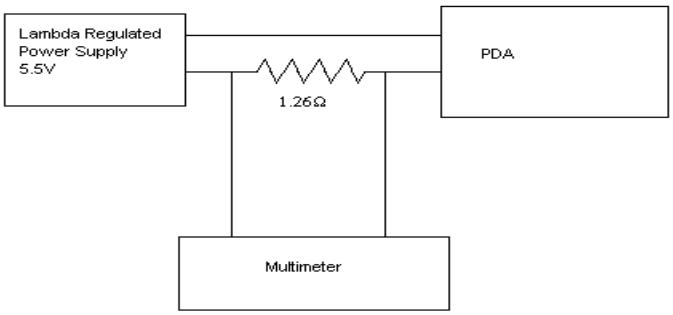
Due to the mobile nature of PDA devices, they have intrinsic power limitations. The question that therefore arises is, how much do architecture issues affect the power efficiency of these devices. Does running certain applications, or utilizing certain subsystems of the machine or certain parts of the processor more than others cause excessive power drain?

In order to answer questions such as these, we decided to measure the power consumption of PDA's under different circumstances. These tests can be separated into two main groups:

- Testing power drain during the runtime of different benchmarks.
- Testing power drain during concentrated use of selected processor functionalities

By testing these two areas, we can get some feeling for how much of an affect these characteristics can actually have on the power efficiency of PDA architectures. We will also measure the power consumption of the performance metrics that we ran as they are representative of the type of power drain to be expected from user codes.

The methodology that we followed in collecting power consumption data was to power each PDA using the same Lambda regulated power supply (5.5V DC) and to measure the power delivered over that line. We were able to make that measurement by connecting a resistor in series with the PDA and measuring the voltage drop across the resistor as different codes ran. By using the known values for the power supply and the resistor, the measured values for the voltage drop across the resistor, and simple voltage relations($V = IR$, $P = IV$), we were able to calculate a value for static power drain.




### 2.1 ALU code sequence

This code sequence was aimed towards establishing some kind of base level of power consumption with an active processor. By running the processor in a tight loop around several ALU instructions, we were able to make some static power drain measurements.

### 2.2 Memory/cache access

This code sequence was targeted at assessing the performance of the cache and memory systems. Because much of the memory hardware was similar, we thought it would be more relevant to examine the performance of the different caches. To that end, we wrote this code sequence to repeatedly perform loads and stores, while ensuring 100% cache hit. Again, we create a tight loop around this code and measure the static power drain.

### 2.3 Procedure Invocation Overhead

The motivation behind this code sequence came out of an examination of the compiled assembly code for the performance benchmarks that were written for the first part of this paper. An examination of that source code revealed that a large portion of that code was overhead dealing with setting up and cleaning up after procedure calls, system call-outs and shared library call-outs. This makes sense, as those benchmarks dealt heavily with manipulating GUI elements and other system level resources.

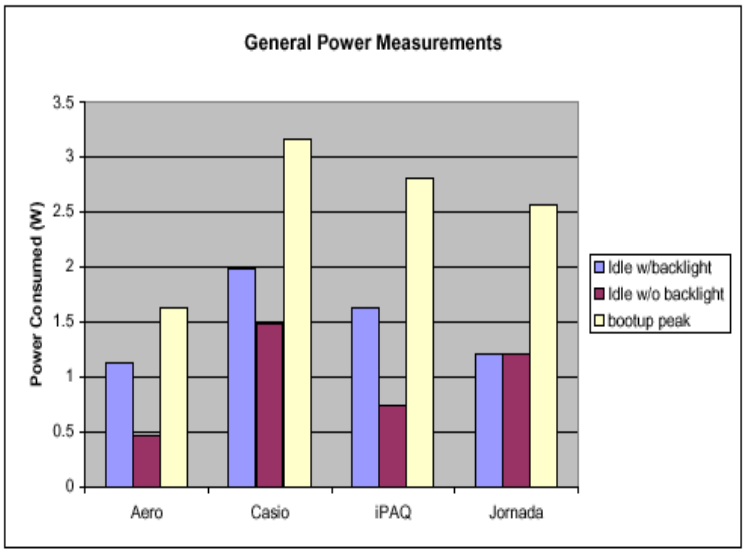
It would then follow that many user applications' codes would follow in suit. Therefore, it seems prudent to investigate how much overhead is introduced by each processor/ISA/calling convention in dealing with procedure invocation. Of the code we examined, we found that the average number of arguments to a procedure invocation was 4 (most likely because many of the procedure calls were to WinAPI functions, many of which take similar arguments) and so we duplicated that in our code.

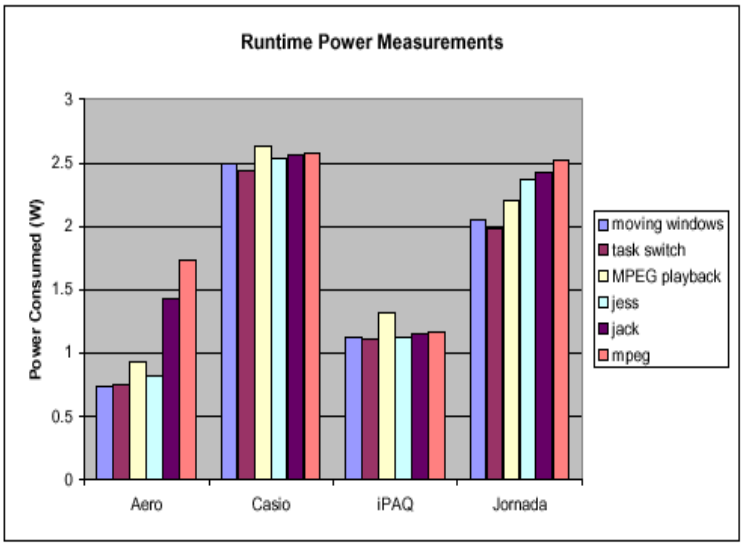
### 2.4 Branch Prediction

The motivation behind this power metric was the differences in the branch prediction schemes between the different processors that we examined. While codes in the previous section of the paper could speak somewhat to the effectiveness of these schemes, it seems pertinent to also examine them for their power implications.

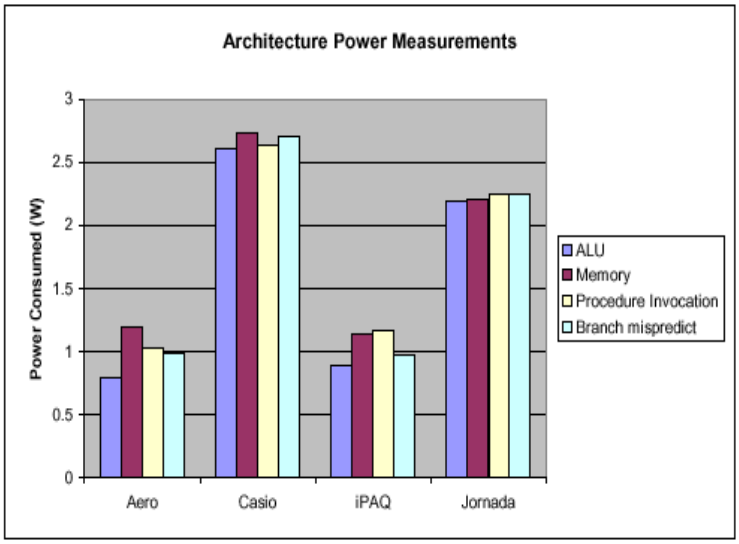
To that end, this set of codes seeks to foul any reasonable branch predictor by randomly selecting from a set of 8 branch targets each time around the loop. We implemented this in the form of a C switch/case statement.

### 2.5 Results






The results of our power measurements did not allow us to draw many in depth conclusions, but at the same time were not at all surprising. We found that, for the most part, all the platforms consumed 50%-75% more power when running applications then when idle. There were no large differences between the individual tests though. The only significant observation that we did make was that the Aero and Jornada spike a bit when running the JVM benchmarks which were memory intensive. Both systems have 16 MB of RAM as opposed to the 32 MB of the iPAQ and Casio. Therefore, we believe that these spikes are most likely due to garbage collection operations in Java. This seems to be supported by SPEC documentation which shows that heap allocation scales directly with our measured values on those two samples.




Looking at the architecture power measurements, we can see that there were no significant differences in

power consumption between different types of instructions. Therefore we can conclude that no single type of operation is noticably less power efficient than another.

Somthing else to note is that from our measurements, the back-light of the various PDAs accounted for a large portion of the power consumption that we measured. Unfortunately, we were unable to completely shut-off the back-lighting on the Casio or the Jornada. Even still, we measured the back-light as responsible for up to 50% of power consumption in idle states.

## Conclusions/Discussion

The aim of this paper was to explore the possibility of creating a standard toolkit for benchmarking and evaluating PDA performance and to use those tools to evaluate currently available PDA hardware samples. The approach to move away from standard scientific benchmark sets and to more accurately simulate real user code was valid and promising, as was the goal of measuring power consumption and tying those results into architectural details and weaknesses.

We were able to develop several good examples of the sorts of generic cross-platform metrics that we originally set out to do. By taking the benchmarks developed here as an example, one could easily expand on what we already have to create a truly comprehensive benchmark to be applied across Windows CE handhelds.

We were also able to take many power measurements of the PDAs as they were running, as we had planned. However, unfortunately, due to logistics and the limitations of what hardware was available to us we feel that the number of variables between the machines that we were measuring were too great to be able to draw any strong, inter-platform conclusions. This was also due, in no small part, to the lack of technical information available from the manufacturers of our samples.

However, we feel that with the toolset that we have developed, and the methodology outlined above and adequate hardware samples, that one can begin to tackle the questions that we set forth to answer.

## Related Work

There has been little work into benchmarking PDAs. The performance studies that have been carried out to date have been case studies focusing on a particular architecture and testing its performance and/or power consumption. By standardizing our codes, we seek to be able to use our results to do a comparative analysis across various architectures.

Conclusions drawn from previous work [HN98] suggests that traditional benchmarks do not accurately capture PDA performance. Furthermore, issues are also raised with the feasibility of accurately measuring power consumption, especially at the granularity required to identify bursty spikes in usage.

Lorch and Smith [LS96] explore the power consumption effects of implementing inactivity power-downs and power-ups. The same methodology can be applied in examining how our different samples deal with the inactivity operations of the Win CE OS.

# Bibliography

[HN98] Jason Hong and Mark Newman. "A Look at Power Consumption and Performance on the 3Com Palm Pilot" 1998

[KS97] Randy Katz and Mark Stemm. "Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices" 1997

[LS96] Jacob Lorch and Alan Jay Smith. "Reducing Power Consumption by Improving Processor Time Management in a Single-User Operating System" 1996

[LS98] Jacob Lorch and Alan Jay Smith. "Software Strategies for Portable Computer Energy Management" 1998

[ARM] Intel Corporation. StrongARM Processor Documentation
http://developer.intel.com/design/strong/sa1100.htm

[SH3] Hitachi Corporation. SH3 Processor Documentation
http://semiconductor.hitachi.com/superh/

[MIPS] MIPS Technologies Corp. MIPS Processor Documentation
http://www.mips.com