Power and Energy Code Profiling in Pharo

Alexandre Bergel
Pleiad Lab, DCC, University of Chile

Abstract

Modeling energy consumption is a major concern when designing software systems such as a cloud service or for a mobile device (*e.g.,* smartphone, laptop). Unfortunately, we have little knowledge on how decisions taken by a programmer may impact the energy profile of a system.

This paper investigates the energy consumption in the Pharo programming language. We have closely monitored the power consumption of five micro-benchmarks and four macro-benchmarks. Our findings indicate that the way the memory is used has a significant impact on the power consumption. We propose a code profiler to measure the power consumption of any Pharo code expression. Our profiler is available under the MIT license and run on Pharo 5.

1. Introduction

Energy consumption is an important aspect when designing software systems. A significant effort is dedicated by the industry to reducing the energy consumption of cloud infrastructures and mobile devices.

The importance of energy consumption is well known, and various energy models have been proposed to restrict or predict energy consumption [1]. Unfortunately, there is little understanding on how the decisions taken by a programmer impact the overall energy consumption. Most of the current approaches are either specific to a particular material or ad-hoc to a particular application. Moreover, the direct impact on the programming activity on the energy consumption is still unknown.

Energy consumption. This paper proposes a technique to monitor the energy consumption in the Pharo programming language. We have run our profiler across a set of nine benchmarks and made the following findings:

- We characterize some memory usages with respect to the memory consumption. In particular, we found that allocating and filling large memory chunks has a lower energy footprint than allocating and filling small memory chunks.
- A computation making a heavy numerical manipulation has a lower energy consumption.

Monitoring the energy consumption is known to be highly prone to unexpected variations. We present the results we measured on a particular hardware using a well identified set of benchmarks. We do not claim to have generalized our results.

Contributions. This paper makes the following contributions:

- It carefully analyzes the energy consumption for some programming idioms in the Pharo programming language.
- It briefly describes the implementation of an energy code profiler for Pharo.
- It presents some of our findings related to energy consumption in Pharo.

Outline. The paper is structured as follows. Section 2 gives the necessary background on power and energy profiling and presents the software and hardware used in our setting. Section 3 describes the power and energy consumption associated to the Pharo virtual machine. Section 4 presents the benchmarks we have used in our experiment. Section 5 details the results of our monitoring using statistical tools. Section 6 briefly describes our energy profiler. Section 7 gives an overview of the related work. Section 8 concludes and exposes our future work.

2. Experiment Design

2.1 Metrics

Before introducing the metrics we will consider in this paper, we briefly recapitulate the units we employ to characterize power and energy consumption.

Energy units. *Joule* (J) is a unit of energy. As a physical representation of this unit, we have the following correspondence: vertically lifting an object of 100 grams by one meter requires an energy of 1 J, from the surface of the Earth.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
IWST '16, Month d–d, 20yy, City, ST, Country.
Copyright © 2014 ACM 978-1-nnnn-nnnn-n/yy/mm. . . $15.00.
http://dx.doi.org/10.1145/nnnnnnn.nnnnnnnReprinted from IWST '16, [Unknown Proceedings], Month d–d, 20yy, City, ST, Country, pp. 1–7.

Watt (W) is a unit of power, defined as joule per second. The Watt unit describes the energy consumption at a given time. The cumulative energy consumption for microprocessors is measured in milliwatt hour (mWh). As a comparison, a typical MacBook Air has a built-in 38 Wh lithium-polymer battery, and a 45 W MagSafe 2 Power Adapter.

Processor energy consumption. Modern processors integrate a graphical unit in addition to the logical unit (*i.e.,* the physical unit that interprets instructions). We define *PkgPower* as the power consumed by the whole processor. This metric is measured in Watts. The power consumed by the logical unit is noted *IAPower*. An Intel i5 processor typically consumes between *PkgPower* = 5 and 40 W, and *IAPower* ranging between 2 and 35 W.

The cumulated processor energy *CuPakPower* is the amount of energy consumed during a period of time, measured in mWh.

Processor Temperature. As far as we know, the Intel i5 does not provide individual unit measurements, *i.e.,* it is not possible to obtain the temperature for the graphical unit and logical unit separately. Instead, the processor exposes the temperature of the whole physical compound. We refer to this global measurement as *PkgTemp*, the temperature of the processor (graphic and logical units).

Temperature of an Intel i5 ranges between 40 and 65 C degrees. All temperature measurements provided in this paper are in degree celsius (C).

Processor Frequency. A processor is associated to a frequency, measured in GHz, reflecting the number of primary instructions the logical unit can execute for unit of time. Modern processors change their frequency based on the current workload. The frequency variation is instructed by the operating system. Consider an Intel i5 having a frequency set of 3.2 GHz. Using OSX 10.11.4, the i5 has its actual frequency ranging from 1.7 Ghz to 3.6 Ghz, depending on the provided workload. Note that the processor may run with a frequency greater than the frequency set by the constructor. This is because Intel processor offers a "turbo mode", which allows overclocking under some particular conditions. The frequency variation usually reflects the energy saving policy provided by an operating system. We denote the metric *IAFrq* the frequency of the logical unit, measured in Hz.

2.2 Software and hardware platforms

We consider the Pharo 5 image and the Cog Virtual Machine[1]. We will run Pharo with OS X *El Capitan*, Version 10.11.4, which is latest version available at the time this paper is written.

We consider the Intel Core i5 available in an iMac (year 2012). The considered i5 has a 3.2 GHz frequency, set by the constructor.

[1] Version released on 4 May 4 2016.

All the measurements were carried out at an atmospherical temperature of 22 C. This is an important factor to consider since most modern hardware has a turbo mode overclocking the processor with a cold atmospherical temperature. Our measurements included a screen using an intensity of 75%. Network connection was disabled during the measurements.

3. Virtual Machine Launch and Being Idle

One should measure the basal energy before carrying out the measurement on a set of benchmarks.

Operating System. We measure the processor energy consumption with no running application. Pharo is therefore not running and all the network connections have been shut down. In that configuration, the processor has a consumption of approximately *PkgPower* = 6 W. The IA unit has a consumption of *IAPower* = 2.5 W on average. The temperature is about *PkgTemp* = 42 C. In the absence of workload, the processor frequency runs with a frequency of *IAFrq* = 1.78 GHz.

Launching the VM. The Pharo virtual machine usually takes less than 1 second to launch on our hardware. The time taken to open the virtual machine is linear to the image file size. Figure 1 represents a typical launch of the virtual machine with an image of 47Mb.

The left-hand side of Figure 1 shows the power consumption at four different times during the virtual machine launch. The right-hand side represents the cumulative energy consumption. Opening a 47Mb image consumes 3 mWh on our platform.

Pharo Idle. Pharo has a very stable power consumption of *PkgPower* = 6 W as soon as no explicit interaction or execution is carried on. It is known that the Pharo VM checks for event every millisecond, leading to a constant little use of the CPU. Despite this, we could not measure any impact on the energy consumption due to this regular event check.

4. Benchmarks

To characterize the power and energy consumption of a processor, we will consider a set of 9 benchmarks: 5 micro-benchmarks and 4 macro-benchmarks.

4.1 Micro-Benchmarks

A micro-benchmark is a very specific workload intended to measure one particular aspect of the processor. We consider five micro-benchmarks:

- *G1*: Creation and elimination of memory blocks large of 3 Mb, represented as an `Array`.
- *G2*: G1 and the memory space are sequentially filled with an arbitrary immediate value.
- *G3*: Creation and elimination of small memory blocks, large of 10 Kb, represented as an `Array`.
- *G4*: G3 and the memory space are sequentially filled with an arbitrary immediate value.

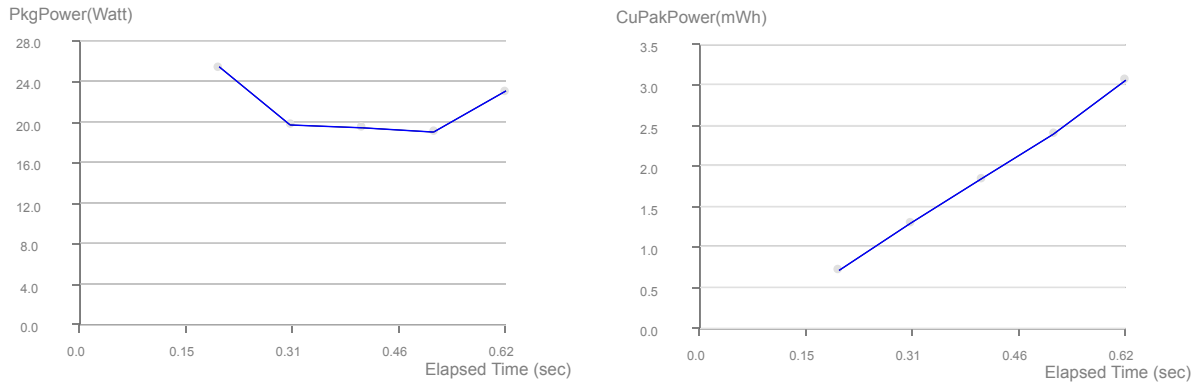

Figure 1: Energy profile of a typical Pharo VM launch

- *Rec*: Recursively computing a large numerical sequence.

The first four micro-benchmarks exercises the garbage collections and memory accesses. All the memory accesses are sequentially performed. The fifth micro-benchmark intensively uses the method call stack.

4.2 Macro-Benchmarks

We designed four macro-benchmarks:

- *FB*: Computing a force based layout on a graph made of 5,000 nodes and 5,000 edges.
- *Graph*: Constructing a graph large 70,000 nodes and 70,000 edges.
- *Plot*: Plotting 30,000 numerical values.
- *Comp*: Compiling over 3,800 Pharo methods.

These macro-benchmarks have been designed to not rely on some external devices such as the network or the hard disk.

4.3 Benchmark Execution

We will run each benchmark across our set of platforms. Between each benchmark execution, we pause Pharo for 10 seconds. This pause ensures that the CPU cools down and reaches a still physical state before the next benchmark execution.

5. Profile Across Different Workloads

We ran our set of benchmarks on an iMac Intel Core i5 3.2 GHz. We were careful in reducing and avoiding possible bias: Pharo is the only application manually launched on our hardware; network connection has been disabled; tests have been multiply run before obtaining our measurements; the screen intensity has been constant and no screensaver has been employed.

5.1 Statistical tests

A data set can be "normally" distributed or not. Being normal has an impact on the statistical tool used to analyze this data. In our case, none of the benchmarks have produced a statistically normal data set. We used the Shapiro-Wilk test to verify normality.

	mean	stdev	median
GC1	18.12	0.3521	18.08
GC2	18.17	0.5578	18.11
GC3	18.23	0.6131	18.10
GC4	19.12	3.975	17.92
Rec	18.25	0.5355	18.25

Figure 2: Micro Benchmark - Description

As a consequence, we use the Kruskal-Wallis test, a statistical test to analyze differences in median values between samples [2]. It is non-parametric (implying that it operates on data that are not normal as we have) and evaluates one particular factor.

In cases where we had more than two data sets to compare, we used the Dunn's multiple comparisons tests.

5.2 Micro-Benchmarks

In total, the five micro-benchmarks produce 5,165 data points. We first analyze the power consumption and then the temperature.

Power consumption. Table 2 gives some statistical measurements. All the micro-benchmarks have an average comparable consumption. The benchmark GC4 clearly stands out when compared with other benchmarks.

Figure 3 charts the evolution of the PkgPower metrics along the time. GC4 is characterized by many sporadic power consumption peaks. Currently, we cannot explain the cause of these peaks. Our measurement indicate that they are not related to the use of garbage collector. GC3 and GC4 produce exactly the same number of incremental and full garbage collections, and GC3 does not present these peaks. We therefore infer that these peaks may be caused by the processor low-level caches [1].

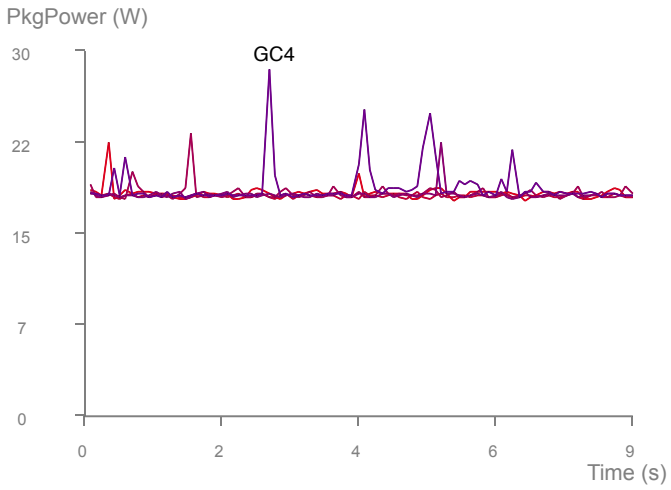

Figure 3: Evolution of PkgPower along the time

Figure 4 gives the distributions of the data points. We see that the median for each benchmark is very similar, around 18W. Each benchmark produces data points within a particular range. The Rec benchmark produces the narrowest range. The reason stems from the low activity of the garbage collector: using the runtime stack does not directly trigger the garbage collector since no heap allocation occurs.

Benchmark GC1 creates and immediately frees large memory portions, thus triggering the garbage collector. Benchmark GC3 creates and frees smaller memory chunks, which then trigger the garbage collector more often than with GC1. This explains why the range of power consumption is larger with GC3 than with GC1.

Sequentially filling memory with an immediate value increases the variability in the power consumption. Considering smaller chunks leads to more variability than larger chunks.

Table 5 gives the result of the statistical multiple comparison. The table indicates whether two data point sets are significantly different. From the table, we can conclude the following:

- On average, multiply allocating large memory chunks consume more power than allocating small memory chunks (GC1 is significantly different from GC3).

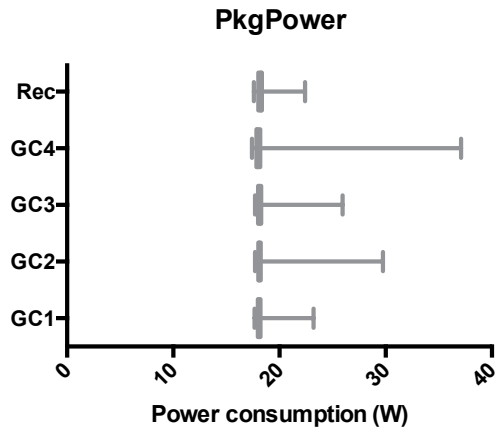

Figure 4: CPU power consumption distributions across micro benchmarks

	GC2	GC3	GC4	Rec
GC1	yes	yes	yes	yes
GC2		no	yes	yes
GC3			yes	yes
GC4				no

Figure 5: Micro Benchmark - PkgPower - significant difference

- Sequentially filling a memory portion has a memory cost (GC2 is significantly different from GC1, and GC4 sig. diff from GC3).
- Using the runtime call stack has a lower power consumption than using the heap (Rec is significantly different from GC1). We conjecture that this is due to the garbage collector.

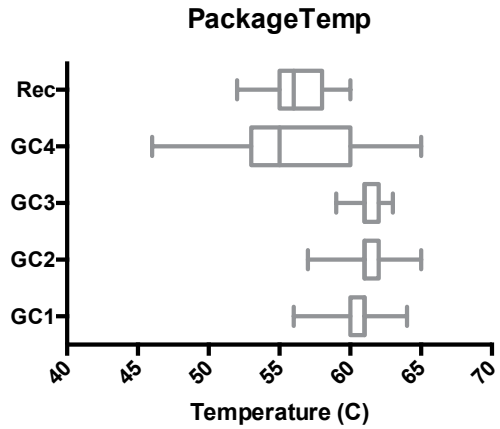

Figure 6: CPU temperature distributions across micro benchmarks

Temperature. Figure 6 indicates the distribution of the temperature of the benchmarks. We can notice that GC4 has a temperature profile different from the other benchmarks. The large temperature range of GC4 seems to be due to its particular power consumption.

5.3 Macro-Benchmarks

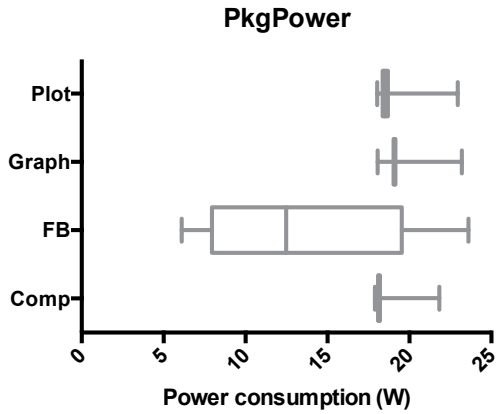

Figure 7: CPU power consumption distributions across macro benchmarks

Power consumption. Figure 7 gives the distribution of the power consumption of our four macro-benchmarks. All but one benchmark have a comparable range of data points. Interestingly, the FB benchmark has the largest range of power consumption. The reason why FB has such a large range is not clear to us. We conjecture that this is because FB makes heavy use of arithmetic operations.

Figure 8 plots the *PkgPower* along the time. After five seconds, we see that the energy consumption of the FB benchmarks drastically drop. The reason for this is not completely clear to us. We suspect that the method cache and the proximity of the data (since they are immediate values) play a large role in this consumption drop.

Table 9 characterizes the differences between the four benchmarks. All the benchmarks have a significant difference between each other, except Comp and FB.

This measurement strongly indicates that each execution has a particular profile. Moreover, intensive numerical operations seem to result in a lower consumption.

CPU frequency. During our measurement, the CPU frequency, *IAFrq*, ranges from 3.4 GHz and 3.6 GHz. It therefore goes above the 3.2 GHz set by the constructor, indicating an overclocking. Our measurements indicates a value *IAFrq* = 3500, sporadically be 3400 or 3600. The fact that *IAFrq* takes only three different values during the execution discards any correlation between the CPU frequency and the power / energy consumption.

Temperature and power consumption. Figure 10 plots the data points for two benchmarks using the metrics *PkgTemp*

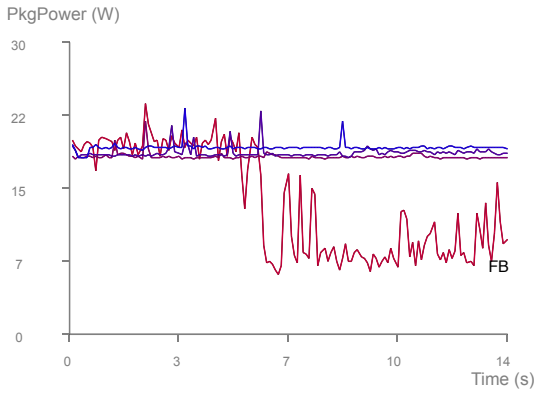

Figure 8: Evolution of PkgPower along the time for the Macro-benchmarks

	FB	Graph	Plot
Comp	no	yes	yes
FB		yes	yes
Graph			yes

Figure 9: Macro Benchmark - PkgPower - significant difference

and *PkgPower*. All the micro and macro benchmarks have a shape similar to Plot. Only FB has its measurement points horizontally stretched out.

We conclude there is no correlation between *PkgTemp* and *PkgPower*, despite a relatively high correlation for one benchmark.

6. Implementation

Simple API. We provide a class `EnergyProfiler` and a method `profile:` to perform the measurement. Consider the following code:

```
EnergyProfiler new
  profile: [ "Block to be monitored" ]
```

The method `profile:` is executed as follows:

1. it launches the low level energy monitor
2. it then evaluates the provided block

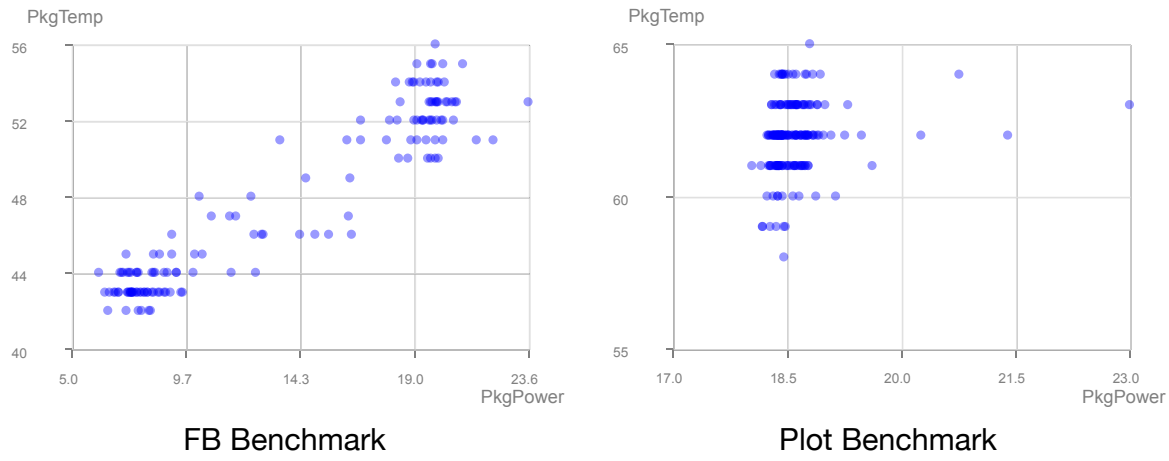

Figure 10: Plotting *(PkgPower,PkgTemp)* for two benchmarks.

3. after the execution of the block, the monitor is stopped, and data are made available in the Pharo image

`EnergyProfiler` also supports exporting the data in a CSV format. This is convenient when data have to be statistically analyzed.

Low-level measurements. All the low-level measurements are performed using the Intel Power Gadget, a tool available for Intel processors[2]. Thanks to OSSubprocess[3], we were able to launch the Intel toolchain within Pharo.

7. Related Work

Kansal *et al.* [1] have described a power consumption in which the power consumed by the memory is linear to the last level cache (LLC). LLC is a memory cache maintained in the CPU that is shared among all processor cores. Unfortunately, the state of the LLC cannot be inspected since the processor does not expose it.

Kistowski *et al.* [3] have shown that the same workload may result in very different energy profiles. They have considered 5 macro-benchmarks and 3 different hardwares. Their results shows that the energy profile of a workload significantly varies across physical processors having the same characteristics. This is a relevant finding for our setting. We have currently run our experiment on a single hardware and software set. As future work, we will run our benchmarks across several executing platforms.

8. Conclusion and Future Work

To the best of our knowledge, the impact of the Smalltalk execution on the power and energy consumption has not been considered. This short paper presents our preliminary results in measuring the energy consumption in Pharo. Our measurements indicate that activating the garbage collector greatly contributes to varying the power consumption.

As a future work, we plan to investigate the following points:

- *Replication across processors*: The work presented in this paper considers one particular processor. It is known that the same workload may have different energy profiles across the same processor [3]. One relevant point to research is whether the virtual machine or the considered version of Pharo have particularities energy-wise.
- *Page swapping*: Paging is a memory management technique offered by most operating systems to move a memory page from the physical memory to a secondary storage. Loading and unloading pages from the hard disk may have a significant impact on the overall consumption. We will therefore explore the use of page swapping.
- *Processor internal caches*: a processor has several memory caches. These caches are hierarchically structured around each processor core. It has been argued that memory caches may have a linear impact on the processor power consumption [1]. We will research how this aspect is exploited in our profiler.

Acknowledgments. We thank Renato Cerro for their comments on an early draft of this paper.

[2] `https://software.intel.com/en-us/articles/intel-power-gadget-20`

[3] `https://github.com/marianopeck/OSSubprocess`

References

[1] A. Kansal, F. Zhao, J. Liu, N. Kothari, A. A. Bhattacharya, Virtual machine power metering and provisioning, in: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, ACM, New York, NY, USA, 2010, pp. 39–50. doi:10.1145/1807128.1807136.
URL http://doi.acm.org/10.1145/1807128.1807136

[2] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Kluwer Academic Publishers, 2000. doi:10.1007/978-3-642-29044-2.

[3] J. von Kistowski, H. Block, J. Beckett, C. Spradling, K.-D. Lange, S. Kounev, Variations in cpu power consumption, in: Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering, ICPE '16, ACM, New York, NY, USA, 2016, pp. 147–158. doi:10.1145/2851553.2851567. URL http://doi.acm.org/10.1145/2851553.2851567