OpenVMS Technical Journal V16

MIKADO, a Virtual VAX Based on a Real-time Operating System

Robert Boers, CEO and CTO Stromasys SA, Switzerland
(For bio see http://www.stromasys.ch/about-us/board-of-directors)

Introduction

The Virtual Address Extension (VAX) processor family by Digital Equipment Corporation has earned its place in the history of computing. Two decades of development and sales exposed computer users to these well-designed and reliable systems, which were copied in former Eastern Europe as the best architecture available. However, with emerging technologies and the quest for ever faster, smaller and cheaper components, VAX hardware has become obsolete.

While computer hardware will eventually wear out, fall apart, burn up or simply stand in the way, its software can remain to be of significant value to its users. Vendors are of course delighted to sell new systems and might even provide software conversion tools. However, application migration is not trivial and is usually impossible if no source code is available.

Modern computer systems can execute complex tasks much faster than their ancestors of two decades ago. That makes it feasible to design a very precise software model of an older computer system and have it execute the original hardware instructions in software at the same or higher speed than the original. For nearly 15 years Stromasys SA (and its former incarnation as 'Software Resources International') has been building such 'virtual VAX' systems to replace aging hardware under the product name CHARON-VAX. (This technology is called 'processor emulation' in the terminology of the Gartner group).

CHARON-VAX products mostly use Windows as host operating system, as their real-time response is sufficient for most usage. Since the virtual VAX implementation provides a direct connection between the original VMS drivers and the peripheral hardware, we found that we could disable any interfering Windows services that would impact the reliability of the OpenVMS operating system, leaving windows to provide boot and file services.

The phenomenal development of computer hardware over the last decade made virtual legacy systems practical: Our first VAX emulator ran on a 300 MHz Celeron, just keeping up with a MicroVAX 3600. Today, on servers with a 3+ GHz CPU clock, the performance of a virtual VAX is 4 to over 100 times higher than the original hardware, depending on the virtual VAX model. On the I/O side, the higher system clock and the faster I/O subsystems of modern servers have reduced I/O latency and increased VMS disk I/O. A virtual MicroVAX disk controller can see file I/O increase by an order of magnitude.

There are situations however, where replicating precise application timing is more important than throughput when substituting VAX hardware with virtual VAX systems. Typical cases are industrial control systems and military applications. The presence of time critical I/O prevented replacement of such systems. We developed several techniques to adjust timing, so that also these systems can be candidates for replacement with a virtual system if a complete redesign and testing is to be avoided. The MIKADO project was one of these investigations.

MIKADO

In 2004 we designed the MIKADO system, a software model of a MicroVAX system that executes a MicroVAX hardware configuration in a QNX Neutrino real-time operating system (RTOS) running on I86 hardware[1]). MIKADO was a port of an existing CHARON-VAX Linux code base, and we retained the existing control and configuration components for convenience. The real-time environment promised a much lower inter-process latency compared to the then current Windows based systems.

In the early Windows based systems we used configurable timing parameters to adjust I/O response. With dedicated hardware (for example, Qbus adapters) and/or careful tuning we could replace VAX systems with special I/O requirements, at the cost of lengthy testing. The purpose of the MIKADO

[1] The name 'MIKADO' was derived from the pick-up stick game. The design goal was that the manipulation of one component of MIKADO would not influence the timing of the other parts.

project was to understand the advantages of using an RTOS compared to Linux or Windows in building virtual hardware products. The prototype resulted – after a year of testing and modifications in a product that is still sold under the name 'FutureVAX'. FutureVAX is typically used in situations where a mixture of simulated and physical legacy peripherals require a specific, virtual system behavior that can be calibrated automatically.

It is interesting to see that at the time of writing this paper (2010) both emulator technology and host system performance have improved substantially, so that Windows or Linux based virtual VAX and Alpha products normally work 'out of the box'. In general, the interest in 'virtualizing' the legacy system is as much about preserving the OpenVMS environment on a standard server environment, as about removing the dependency on old hardware and avoiding application migration.

However, it does not mean that an RTOS based virtual legacy system has no role anymore. There are situations where an accurate replication of the original system performance and timing is required. Experience has shown that process control or military legacy systems were often 'tuned' to fix problems quickly, at the expense of proper code design. Operating system patches, adding NO-OPs to make custom drivers work or padding serial output with blanks to avoid implementing a handshaking protocol are difficult to fix if only the binary application code remains. While you can argue about the correctness of such implementations, as with all other legacy hardware, such systems will eventually have to be replaced.

The MIKADO advanced development project investigated several ways to calibrate application timing on virtual systems in a way that was independent of the performance of the host system. We had prior experience in system/application calibration with a virtual PDP-11 system where the user could specify the execution time of each individual instruction in any of its major modes. It worked but was complex to calibrate; the impact of individual instruction timing changes on a specific application is usually unpredictable.

In our experiments we found that, given enough system headroom, the actual abstraction of peripherals in a virtual system is not a problem. As long as the emulated I/O device can respond to the demands of the physical connection (serial bit rate, SCSI clock rate, and so on) in time, its absolute time to execute a command is usually not important. In a well designed virtual system nearly all the peripheral abstraction layers (with very few exotic exceptions) cause a minimal host system load.

The element determining the overall virtual system performance is normally the virtual CPU. In the early Windows based systems, the required context switching to serve virtual peripherals contributes to the system overhead and timing inaccuracies. At that time, MIKADO provided a much more predictable solution for I/O timing. However, the emergence of hyper threading, and later multicore systems largely removed that problem, at least for virtual systems with a single CPU.

The next step was to define a way to influence and measure I/O latency. For VAX CPU performance there is the traditional VAX 'VUPs' calculation. There is no established way to compare I/O latency between hardware and virtual systems. One of the experiments concerned an 'oscillator', using a parallel I/O card (DRV11J) with one output connected to an input. The output change triggered an input interrupt that changed the output again, using a standard VMS parallel I/O driver and a (very short) applications program. We compared a hardware MicroVAX 3600 with an virtual VAX, using in the latter case a Qbus adapter (BCI 2100 from The Logical Company) to connect the DRV11J. We did similar tests via (emulated) printer ports.

Based on several experiments we concluded that the simplest way to adjust system performance is to suspend the virtual CPU for a variable time. The exact point where and how to suspend CPU performance required a great deal of thought and experiments; a good implementation (for example, handle pending interrupts first) determines the ultimate virtual system quality.

The CHARON-VAX code is written in C++, and did not pose a problem porting to QNX. The internal messaging structure was partly changed to use QNX specific features. Most of the code modifications

took place in the virtual VAX CPU. The product licensing mechanism (a physical key) required a complete re-implementation as the product we used was not supported in an RTOS, but that part did not influence the virtual VAX functionality. The work resulted in a well tunable virtual VAX system on QNX, but we were still missing the tools to easily tune the performance and automate the process, if desired.

Incidentally, where in our virtual PDP-11 the extra microsecond delays in instruction execution were too short for a context switch, a suspension of a few milliseconds in the right moment allows the host CPU to attend to other tasks, or offload the host system. Without specific measures, the virtual VAX CPU will run the VMS idle loop, loading the host CPU to 100% whenever it gets CPU cycles available. The work on calibration software provided a means to detect the VMS idle loop and suspend virtual CPU operation. We put these findings back into our Windows and Linux based products. This provides the feature that running a lightly loaded VMS environment using a virtual VAX or Alpha on a laptop will not drain the laptop battery at maximum speed.

As a result of the MIKADO project, we developed the FutureVAX product (a QNX based virtual VAX), which supported replacement Qbus hardware and an embedded PC on a quad Qbus card form factor, designed by The Logical Company. The latter is essentially a plug-and-play solution to replace most of the components of a MicroVAX, but retain the existing enclosure and any special I/O hardware.

Tools for calibrating virtual systems

The final phase of the MIKADO project was to build an easily usable calibration tool, including the possibility for the virtual system to calibrate itself. That means that the virtual system can measure its own performance during run-time and can change its emulated system performance accordingly. In this way, a virtual system and its applications could change to a different host platform (including a VMware client) reset itself to the desired performance. Note that automatic calibration is currently rarely used, but we plan to use it in future products for the industrial automation market.
Automatic calibration raises an interesting dilemma. A virtual VAX is designed as an exact replica of the original hardware. The CHARON-VAX management console, which controls proper operation of the emulated environment, sits on the 'other side' of the virtual layer, and is not visible from VMS or its applications. But it is the management console that has to change some component parameters to change the virtual VAX performance dynamically. This requires a communication channel across the virtual system layer.

There are several ways to create such a communication channel. One way is to create a special peripheral (for instance a serial port) that does not connect to the outside world but into the CHARON management console. However, such a solution could interfere with an existing application and is guest system specific (we want to use the same mechanism for other virtual platforms). The solution we chose was adding extra architectural registers to the virtual CPU, which are obviously not used by any software coming from a physical VAX. A small OpenVMS software utility (see appendix) provides the connection to the CHARON-VAX management console.

The selected unit of delay in the virtual VAX CPU is long enough to allow a context switch in Windows. In QNX the context switch is much faster, but we wanted the test package have comparable characteristics on all platforms. As the appendix shows, by increasing the number of delay units we obtain a virtual system performance curve. We found that the 'VUPs curve' and the frequency of our 'oscillator test' followed approximately the same curve, leading to the assumption, that carefully slowing down the virtual CPU is a workable method to adjust critical I/O timing. Testing with some known difficult applications confirmed this.

The slowdown facility is integrated in many of the CHARON-VAX products. The MIKADO package allows the slow down factor to be set in static mode (in the CHARON-VAX configuration file) or dynamically (by calling SLOWDOWN or SPEEDUP from an OpenVMS application). And our current

products also inherited the power save mode when the guest operating system on our virtual platform is idle (for those operating systems where we can detect an idle mode) and a facility for the emulated VAX to shut down its host system if all the work is done. These features reduce system load on (blade) servers as well.

Thanks to their unsurpassed reliability, there are still many VAX VMS systems in the world that control power plants and distribution networks, railroads, steel and automobile plants, military test systems, and other applications with strict real time requirements. Unlike administrative systems, their connection to other equipment made replacement with virtual VAX platforms more complex. However, after more than 30 years hardware parts become scarce, and often the knowledge about these critical systems is no longer present in a company. We now see an increasing demand for replacing such VAX and even PDP11 systems, probably the tail end of the installed base. Fortunately, the power of modern servers and a better understanding of how to virtualize such systems provide the means to preserve these software investments across hardware generations.

Appendix: The MIKADO CPU calibration package

The MIKADO CPU calibration package is a collection of utilities to influence the execution speed of the virtual VAX in a precise manner. While in general the fastest possible code execution is desirable, the implementation of real-time systems can require a specific VAX performance. Host hardware varies widely in execution speed, and setting performance will usually require the combination of an emulator calibration phase followed by a programmed slowdown.

When the host system resource use must be managed, it is useful to reduce the emulator load on the host system when the guest OS (usually VAX/VMS) is idle. It can also be desirable to terminate virtual VAX instance after the guest OS terminates, for instance to release the MIKADO disk images for automated backup. To permit calibration for individual VAX/VMS applications and instances, the performance control can be handled from within the VMS environment by means of VAX/VMS utilities which directly interact through the emulated VAX system with the MIKADO management system.

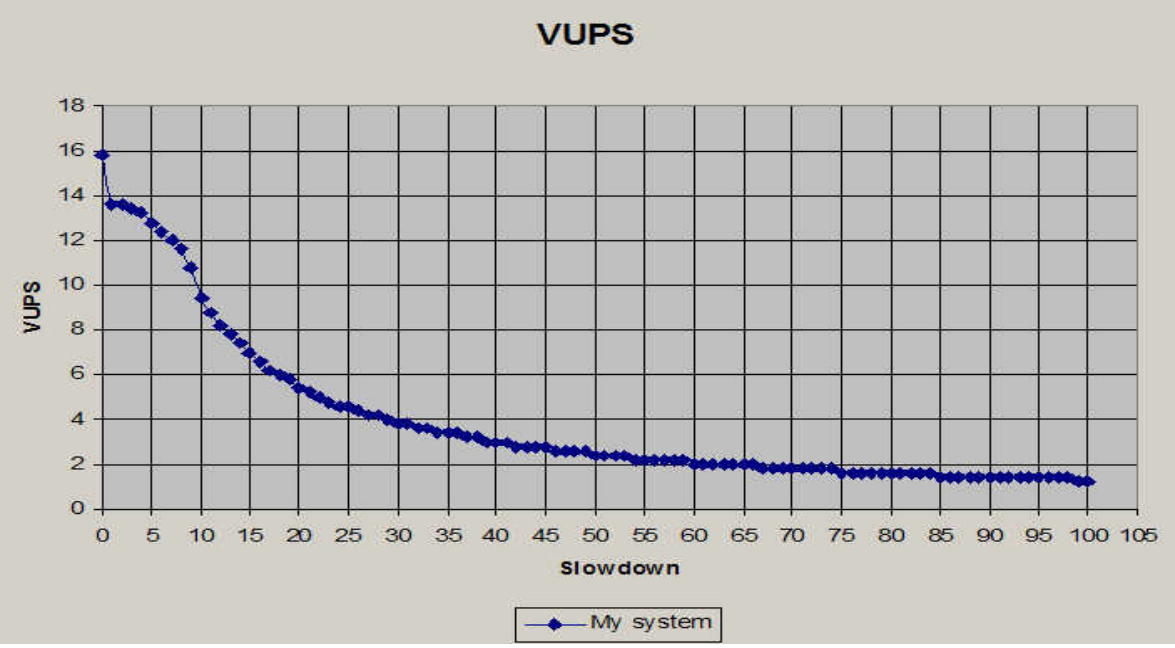
The MIKADO calibration package provides the following performance calibration functions, running as OpenVMS applications on the virtual VAX:

CALCULATE_VUPS.COM	VUPS meter, used with the SLOW_TEST system calibration mechanism.
SLOWDOWN.EXE	Slow the emulator one step.
SLOWDOWN_R.EXE	Remove the execution slowdown – run in full speed.
SLOW_TEST.COM	Calibrates the slowdown mechanism.
SPEEDUP.EXE	Increase the emulator speed one step.

The actual relative system performance can be measured by executing SLOW_TEST.EXE at the VMS prompt (SLOW_TEST.EXE uses the CALCULATE_VUPS.EXE utility). The generated slow_test.log file expresses the VUPs performance in terms of slowdown 'steps', which are reductions of the virtual CPU clock, where the value 0 corresponds to unmodified performance:

```
Calculating at slowdown 1...
   Approximate System VUPs Rating :  13.4
Calculating at slowdown 2...
   Approximate System VUPs Rating :  13.2
…
```

The result of the slowdown test lists the relationship between the slowdown value and VUPS value on the virtual VAX, as illustrated in the following example:



The desired slowdown factor can be specified in the MIKADO configuration file. Using the above example, if you want the system to run at a performance of ~ 4 VUPS, you can just add to the configuration file: set idle slowdown=29

The relationship between the slowdown rate and emulator performance is non-linear, and the VUP measurement fluctuations can create apparent small increases in performance when the slowdown mechanism is increased one step. On virtual systems, the VUPs measurement is an approximation with a statistical variation (maximum 5-10%) increasing with a larger emulated memory size. It will be rarely necessary to remove the statistical error, but if needed you can run the calibration multiple times and calculate the standard error at each point.

Assuming that the maximum error is approximately 10% of the value, a 4 VUPS setting will provide 3.6 - 4.4 VUPs. Note that the VUPs value does not indicate I/O performance; it is only a measure of the VAX CPU - memory interaction. The precision of the performance slowdown increases with an increase of the slowdown parameter. This suggests that using a fast host system provides a more precise environment.

Dynamic performance calibration

The MIKADO VAX emulator performance depends on various factors: the host system CPU(s), its memory architecture, use of the host system CPU for other applications, the actual MIKADO configuration, the peripheral interrupts it handles and the VAX operating system (typically VMS) configuration. Consequently, while the approximate emulator performance can be predicted based on the host platform, calibration will be required for a more accurate setting.

With an insertion of a slowdown parameter in the MIKADO configuration file, as described above, this setting will apply to the whole configuration during its total runtime. However, MIKADO provides a real-time connection between the OpenVMS operating system and the emulator component manager. For dynamic performance adjustment call SLOWDOWN.EXE and SPEEDUP.EXE at the VAX/VMS level to set the right performance in any critical step of a workload execution. SLOWDOWN_R.EXE can be dynamically used to remove any slowdown. By including CALCULATE_VUPS.COM (or application specific responses) in the performance management, the guest VAX/VMS OS can auto-calibrate its execution speed at any stage of application execution.

Host system control

The MIKADO calibration package provides the following host system control functions, running as OpenVMS applications on the virtual VAX:

IDLE.EXE	Reduces host system load when the Guest OS is idle.
SHUTDOWN.EXE	Shuts down the emulator in 30 seconds.
SHUTDOWN3.EXE	Shuts down the emulator in 3 minutes.
SHUTDOWN5.EXE	Shuts down the emulator in 5 minutes.
SHUTDOWN_R.EXE	Remove scheduled emulator shutdown.

The IDLE utility is required for all calibration functions, as it provides the connection to the virtual VAX management console.

For more information

www.stromasys.com