White Paper

Differentiating Features of the PERC™ Virtual Machine

By Dr. Kelvin Nilsen,
CTO, Atego

Revised Edition May 18, 2009

Differentiating Features of the PERC™ Virtual Machine

This report helps customers identify key issues pertinent to selection of a virtual machine for use in embedded applications and explains how the PERC virtual machine differs from alternative offerings.

As Java™ technologies migrate into the domain of embedded systems, it is important for the companies who are adopting these technologies to carefully consider their objectives and to evaluate their technology suppliers according to these objectives[1]. Atego's PERC Virtual Machine offers a unique combination of capabilities and features designed to empower developers of complex embedded systems with the benefits of the Java platform.

Unique attributes of the PERC virtual machine include:

- Support for current J2SE libraries rather than limited "personal" or "micro" subsets.
- Industry leading ultra-reliable, real-time garbage collection.
- Transparency APIs and command shells to provide visibility and control over internal operation of the virtual machine.
- Native compiler and operating system ports for key real-time operating systems.
- Predictable real-time operation of threads, synchronization, and timers.
- Desktop-caliber developer tools including symbolic debuggers, run-time performance profilers, and simulators for popular real-time operating systems running on Solaris and Windows.

These features combine to make PERC the preferred virtual machine for software targeted to command and control of defense systems, network infrastructure management plane, and distributed real-time control of industrial automation equipment.

1.PERC is a trademark of Atego. Java is a trademark of Sun Microsystems, Inc.

Philosophical Distinctions

The Java brand name means different things to different people. The breadth of technologies spanned by this brand name range from "micro" JavaCard applications running in tens of kilobytes to large server applications running with the Java 2 Enterprise Edition and requiring multiple gigabytes of memory.

CLEAN-ROOM IMPLEMENTATION

Atego views the Java platform as an ideal technology for a variety of important applications, but does not license the Java trademark or implementations from Sun Microsystems. Instead, Atego has independently developed and/or licensed from independent sources the standard J2SE libraries[1]. In so doing, Atego maintains greater flexibility in how it implements Java-compatible services and greater latitude in how it licenses these technologies to customers. The freedom from Sun Microsystems licensing restrictions allows Atego to respond more quickly and more directly to the special needs of each customer.

Atego concentrates on serving the needs of developer teams who are working on large or complex software systems. Atego views the strengths of the Java programming environment as being most relevant to applications that face challenges in size or complexity. Examples of the sorts of challenges that motivate selection of Java over more traditional technologies such as C and C++ include:

1. Network-centric and distributed applications, especially distributed applications that span multiple different kinds of computing devices.

2. Programs assembled of components that total more than 50,000 lines of source code.

3. Programs assembled from independently developed and maintained software components, including layered architectures such as network infrastructure in which independent groups take responsibility for data-, control-, and management-plane software.

4. Programs maintained by multiple individuals or teams, over multiple years.

5. Programs with workloads that exhibit highly dynamic behavior, characterized by processing phase changes and/or variable data set sizes with memory serving different purposes at different times.

1. With Sun Microsystems' recent release of open source standard edition Java libraries, the PERC Ultra virtual machine now bundles certain libraries licensed from the OpenJDK project.

6. Programs that require coordination with other computers and/or devices, especially if interactions are asynchronous.

7. Programs that exhibit considerable evolution, such as rapid feature expansion.

8. Programs that require considerable extensibility or customization, either on behalf of the end-user customer or by the end-user customer.

Certain Java initiatives (e.g. J2ME, CDC, CLDC, MIDP) are targeted to market opportunities that may not benefit from the strengths that make Java appropriate for programming in the large. Atego does not target these sorts of applications. Instead, Atego concentrates on serving the needs of developer teams who are working on large or complex software systems. Representative applications include the network infrastructure management plane, defense system command and control, and distributed real-time automation.

DEEPLY EMBEDDED

Atego concentrates its efforts on supporting mission-critical infrastructure applications. In general, these applications serve quietly and reliably out of human view, interacting with the real world by communicating with other computers and devices over network connections.

Because these applications serve mission critical roles without direct human interaction or monitoring, it is essential that they function reliably and autonomously. Often, the need to support autonomous response to faults and other unpredicted events is part of what adds to the complexity of the software. Unlike virtual machines offered by competitors, the PERC virtual machine is carefully engineered to support ultra reliability without compromises.

COMPANY HISTORY

Dr. Kelvin Nilsen's Ph.D. research focused on high-level programming language features for real-time programming. Real-time garbage collection was one of the topics addressed in his 1988 dissertation. After Dr. Nilsen joined the faculty of Iowa State University, he received research funding from the National Science Foundation and the U.S. Department of Commerce to continue his research on these topics. By 1995, when the Java programming language was first announced by Sun Microsystems, Dr. Nilsen's research had matured to the point of commercialization. In 1996, Dr. Nilsen left Iowa State University to found NewMonics. NewMonics negotiated exclusive world-wide

license rights to all of the source code and the six patents that Dr. Nilsen developed while at Iowa State University.

Early NewMonics customers included DARPA and Rockwell-Collins. These early customers helped establish strong traditions of technology innovation and rigor in software process.

Since the beginning, NewMonics' focus was on the needs of developers who were building complex embedded systems, many, but not all of which, had at least some real-time requirements. Common requirements shared between certain market segments as matched to the unique capabilities of the PERC virtual machine allowed narrowing the focus to particular strategic market segments. Within these targeted market segments, customer feedback has provided guidance and opportunity to further improve the match between the needs of these targeted customers and the capabilities of our products.

In 2005, Atego, an established supplier of Ada programming technologies for mission- and safety-critical systems, acquired the IP assets and employees of NewMonics.

REPRESENTATIVE DESIGN WINS

Since acquiring NewMonics, Atego has oriented its development and marketing efforts towards four key market segments: network infrastructure, industrial automation, defense command and control, and fleet telematics.

- Network infrastructure includes optical platforms, multi-service access devices, service concentration equipment, and IP infrastructure. Each of these so-called "network elements" contains large amounts of software. The lowest-layer software has little if any dynamic functionality and little need for evolution of functionality, little if any differentiation between vendors, and demanding performance requirements. This low-level software is almost always written in C. Higher-level management-layer software has considerable complexity, evolves rapidly, and represents the majority of differentiation between services offered in competing products. PERC technologies are ideal for this management layer functionality. The industry leading Calix C7 multiservice access platform is built on the strong foundation of the PERC Ultra virtual machine.

- Industrial automation consists of using computers to run manufacturing machinery and test equipment. The benefits of PERC are most relevant in applications that involve flexible manufacturing (for which the same manufacturing equipment is used to produce different products at different times) and distributed real-time control

systems, in which multiple devices (robot arms, assembly line, quality assurance inspections) need to coordinate their efforts within timing constraints. National Oil Well Varco has developed the automation software for an off-shore drilling rig in the Java language, deployed on the PERC Ultra virtual machine. The system reliably interleaves the execution of real-time garbage collection with the execution of various real-time tasks which execute periodically with a 10 ms period.

- Defense system command and control spans the domains of network infrastructure and industrial automation. Because of the hostile environments within which they operate, defense system networks are highly dynamic, including the capability to heal themselves whenever certain nodes or network communication channels are destroyed. Security concerns are very high. And automation plays critical roles in reducing the costs of military engagements and removing personnel from harm's way. Recent examples of defense system applications built with the PERC Ultra virtual machine include Aegis warship modernization, Future Combat Systems Systems of Systems Common Operating Environment, and the UK's Taranis unmanned aircraft.

- Fleet telematics describes computer equipment installed in automobiles for purposes of assisting with navigation, generating regulatory reports, communicating information between drivers and central dispatch, and tracking vehicle and driver performance. PERC software is ideal for these sorts of applications because the systems must support multiple functions, the features of each function are continually evolving, and many such systems require software upgrades to be automatically installed in field-deployed systems. Xata Corporation's Xatanet product, running on Symbol and Qualcomm equipment, is an example of a fleet telematics application that depends on the unique capabilities of the PERC virtual machine.

DEVELOPER TOOLS

Over the years, Atego' PERC product has accumulated a breadth of unique developer tools, making it the most robust virtual machine technology available to the embedded developer. The following list highlights some of the valuable tools:

- Fast Predictable Execution: Optimizing ahead-of-time (AOT) and just-in-time (JIT) compilers for important embedded targets including Pentium, Power PC, XScale, and StrongARM. Many other embedded VMs lack compilers, or only support JIT or AOT, but not both. The JIT and AOT compilers offer significant speedup (up to 25 times) over interpreted execution. The option to use AOT compilation has been critical to telecommunication equipment vendors requiring five "9s" of availability. Their software process requires that they test exactly the same software that they are going to ship. In part, this reduces the likelihood that errors will be introduced by adaptive JIT compilers and optimizers. Additionally, in the event that customers

encounter bugs in the field, this approach makes it much easier to duplicate the customer's environment in the test lab.

- High-Level Debugging: Remote symbolic debugging of interpreted and compiled code using industry-standard JDWP protocols. Many other embedded VMs do not support symbolic debugging or allow symbolic debugging only of interpreted code.

- Desktop Simulation of Embedded Environments: RTOS simulator support, enabling developers to simulate VxWorks environments using VxSim under Windows NT and Solaris. PERC is the only VM to run under the VxSim simulator.

- Performance Tuning: Integrated support for run-time performance profiling based on industry standard JVMPI interface, allowing integration with various Java profiler tools, including Eclipse TPTP. Also provides support for the Java Agent interface and java.lang.instrument package, allowing for custom-tailored instrumentation of byte codes either at run time or ahead-of-time via the ROMizer. This enables tools such as Profiler4J to do customized profiling and performance tuning.

- Static Compilation, Configuration, and Linking: Atego' ROMizer® allows developers to select certain Java libraries to be loaded, resolved, and in some cases compiled to native code prior to deployment on the target system. In essence, PERC tools enable a traditional C-style development approach consisting of edit, compile to native, and link; a traditional Java-style development approach consisting of edit, compile to byte code, and execute; and a hybrid approach in which some components are compiled to native ahead of time and other components are either interpreted or compiled to native just prior to execution (JIT).

- Java/C Integration: Support for Java Native Interface (JNI) and PERC Native Interface (PNI) for integrating Java and C code. The unique PNI offers smaller footprints and higher performance than JNI.

- Java/RTOS Integration: Sandbox configuration options for the virtual machine, allowing developers to limit memory usage and determine the operating system priorities at which the virtual machine threads run.

Library Support

JDK 1.5 STANDARD EDITION

Unlike other embedded virtual machines, most of which focus on achieving the smallest possible memory footprint, the PERC virtual machine is biased towards supporting greater functionality. Instead of supporting the restrictive J2ME or Personal Java library subsets, the PERC virtual machine provides compatibility with J2SE. Examples of standard Java library functionality that is present in J2SE but not in the smaller subsets include java.beans (framework for components), java.rmi (provides remote method invocation services), and java.security (supports encryption and authentication). This makes it much easier for developers to leverage existing Java expertise and integrate off-the-shelf Java software components, such as SNMP, web-based management, and CORBA implementations, into their embedded systems.

PERC Ultra IS NOT THE REAL-TIME SPECIFICATION FOR JAVA

Though PERC Ultra is the most widely deployed real-time virtual machine, PERC Ultra does not implement the Real-Time Specification for Java (RTSJ). The design of PERC Ultra predates the RTSJ by many years. Atego has carefully weighed the benefits and costs associated with supporting the RTSJ specification. So far, our analysis has concluded that the costs far outweigh the benefits. Specific concerns include the following:

1. As a real-time solution for Java, PERC Ultra has a much longer history and a more proven track record. PERC Ultra has been commercially available since 1997. It has been successfully deployed in hundreds of thousands of devices, with many millions of hours of demonstrated availability at five-nines and higher.

2. PERC Ultra is developed and supported by a company that focuses on mission-critical and embedded real-time systems. Atego has a 25-year history of supporting these industries. Customers are universally pleased with the high levels of attention and responsive support dedicated to their special needs.

3. PERC Ultra is based on standard edition Java with special implementation techniques to enable reliable and portable real-time operation. In contrast, RTSJ approaches to real-time Java require developers to use the specialized APIs of the RTSJ to achieve real-time behavior. These specialized RTSJ APIs are less desirable because:

    a. Off-she-shelf Java functionality is not compatible with the RTSJ APIs and must be rewritten in order to achieve real-time behavior.

    b. The RTSJ APIs are very difficult to use and their use is error prone.

c. The RTSJ APIs do not offer the portability, maintainability, and scalability benefits of traditional Java.

d. The RTSJ APIs add considerable complexity to application code and to virtual machine implementation, making it much more difficult to achieve safety and security audits.

e. The RTSJ incurs speed and space overheads that are unacceptable in many real-time systems.

4. In situations that require lower level capabilities than are appropriate with standard edition Java (such as hard real-time, device drivers, interrupt handlers, and safety certification), Atego offers PERC Pico, a product that complements the capabilities of PERC Ultra. A mechanism allows PERC Ultra and PERC Pico to be efficiently and robustly combined in mixed-mode programs that span the spectrum from high levels of abstraction implemented in PERC Ultra to the low levels of concrete detail typically implemented in PERC Pico. PERC Pico is based on JSR-302, which is structured as a subset of full RTSJ. In comparison with traditional RTSJ, the PERC Pico product offers:

   a. Superior portability, maintainability, and scalability benefits.

   b. Much better speed and much smaller memory footprint.

   c. Static analysis of important real-time properties to assure higher levels of software integrity.

VM Management

THE VM MANAGEMENT API

A Java virtual machine is a sophisticated run-time environment, providing many more high-level services than typical minimalist real-time operating systems. In order to provide the full generality required by Atego customers, the PERC virtual machine has even more internal sophistication than other embedded virtual machines. Achieving optimal system performance depends on finding appropriate balances between the memory and CPU-time budgets assigned to application threads and to certain background maintenance activities. By providing APIs to access and control this information, the PERC virtual machine makes it possible for software agents to take responsibility for self configuration of the embedded system. Compare this to the mini- and mainframe computer markets of previous decades, in which full-time staffs were often required to continually monitor and adjust performance parameters, and some companies made

entire business out of providing software tools that assist with this effort.

Understanding what is happening inside the virtual machine, and configuring the virtual machine for optimal performance with a given workload, requires visibility into the internal operation of the virtual machine and requires that software be able to adjust the virtual machine configuration on the fly. The PERC virtual machine's management APIs make this possible. Some of the provided services include:

- Query and modify the maximum number of heap allocation regions.
- Query and modify the rate for expansion of the memory allocation heap.
- Query and modify the frequency and priority at which increments of garbage collection work are performed.
- Determine how much CPU time has been dedicated to execution of particular Java threads.
- Determine which synchronization monitors are locked by particular threads, and which threads are waiting for access to particular synchronization monitors (to enable analysis of deadlock and resource contention bottlenecks).
- Obtain a snapshot of the stack backtrace for a particular thread, to see what code it is executing at any particular moment.
- Determine the number of I/O requests issued by a particular thread.
- Query and control whether eager linking is enabled.
- Query and control whether JIT compilation is enabled.
- Query and control whether byte-code verification is enabled.
- Query the RTOS priority at which PERC threads run. (Override the default value when you start up the PERC virtual machine.)
- Query the duration of a PERC thread's tick period and time-slice duration. (Override the default values when you start up the PERC virtual machine.)
- Determine how much time the PERC virtual machine has been idle, and how much CPU time has been consumed at each priority level (to assist with rate-monotonic scheduling analysis).

THE PERC SHELL

The PERC shell provides a command-line interface to allow human operators to interactively query and control virtual machine behavior. The shell runs in a dumb terminal connected to a serial port or in a telnet window connected by ethernet. Each PERC virtual machine supports multiple shells running concurrently.

The PERC shell provides access to all of the VM management services described above. You can use the shell to list threads and monitors, view lists of open files and sockets, and examine the "lost file list", which identifies files and sockets that were reclaimed by

the garbage collector before they had been closed by the application. Additionally, the shell supports various user conveniences including file system directories, redirection of standard input and output, and wildcard file naming. Users can extend the shell by writing new Java classes.

Memory Management

SUPERIOR AUTOMATIC GARBAGE COLLECTION

One of the high-level productivity aids provided by the Java programming language is garbage collection, a service which automatically reclaims and recycles memory which had been previously allocated to serve a temporary need and is no longer being used by the application.

All commercially available Java virtual machines collect garbage, but some implementations do it better than others. Conservative and partially conservative garbage collectors make conservative estimates of which memory is still being used. This means they cannot guarantee to reclaim all of the dead memory in the system and usually cannot defragment memory by relocating in-use objects to consecutive memory locations. Note that a single dead object that is improperly identified as live may hold references to an arbitrarily large collection of additional dead objects, all of which must be conservatively treated as live. Thus, the memory for these dead objects cannot be reclaimed.

A key point in understanding conservative garbage collection is that the garbage collector is not always able to distinguish between memory locations that contain pointers and those that contain scalar data such as integer or floating point values. When the garbage collector examines these questionable words of memory, it asks itself the question: "If this in-memory value is a pointer, which object would it refer to?" It then treats the referenced object, if there is one, as live. Note that no amount of testing will guarantee that conservative garbage collection will reliably reclaim the dead memory associated with your application. A memory cell that holds a date and time, for example, might consistently be recognized as a non-pointer during testing. However, when the deployed software is exposed to the full range of possible date and time values, it is likely that certain combinations of values will be perceived by the conservative garbage collection system as references to Java objects, and this will cause unwanted retention of unpredictable amounts of dead memory.

Unlike the garbage collection systems used by other Java virtual machines, the PERC virtual machine uses unique patent-protected real-time garbage collection that is accurate, incremental, defragmenting, and paced. This means that (1) there are no risks of running out of memory because of conservative approximations made by less sophisti-

cated garbage collection systems, (2) both performance and reliability of memory allocation benefit from the PERC garbage collector's ability to defragment memory, and (3) a properly configured PERC virtual machine will not experience unpredictable stalls at arbitrary times because of interference from the garbage collector.

Real-Time Operation

REAL-TIME GARBAGE COLLECTION

Because automatic garbage collection reads and writes the same in-memory objects that Java application threads manipulate, it is necessary to carefully coordinate these independent activities. Because of this coordination requirement, it is common for Java threads running on other virtual machines to experience periods of time when they cannot run because the garbage collector is performing certain critical operations that cannot be interrupted. The duration of these garbage collection interruptions depends on the virtual machine architecture and its configuration. Often, the bound on garbage collection delays is proportional to the size of the allocation heap. Users of alternative virtual machines have reported garbage collection delays ranging from 1 second to tens of seconds with real-world applications. Though these interruptions may be rare, the costs of experiencing a long garbage collection delay at an inopportune time may include mild end-user irritation, lost customers, failed transactions, dropped telephone calls, or manufacturing defects. For example, one large electronic banking organization reported that it had experienced tens of seconds of garbage collection delays with an enterprise-class virtual machine just before market trading ended, forcing the bank to sit on millions of dollars worth of transactions over a weekend.

One of the first innovations introduced in the PERC virtual machine is its patent-protected real-time garbage collection system. The PERC garbage collector divides its effort into thousands of small uninterruptible increments of work. Depending on the choice of underlying CPU, the maximum time required to execute an increment of garbage collection is less than 100 µs. When garbage collection resumes following preemption by a higher priority application thread, it resumes where it left off. There is never a need to go back and restart any phase of garbage collection.

Figure 1 illustrates the incremental copying garbage collection technique used within the PERC virtual machine. At the start of garbage collection, from-space contains the three live objects A, B, and C and to-space is empty. Garbage collection consists of incrementally reserving space and subsequently relocating each of the live objects. Any attempt to access the object during garbage collection is automatically redirected to the single valid copy of the object. At the time this snapshot was drawn, the valid versions of objects B and C are B' and C' respectively. The valid version of A is A itself, because

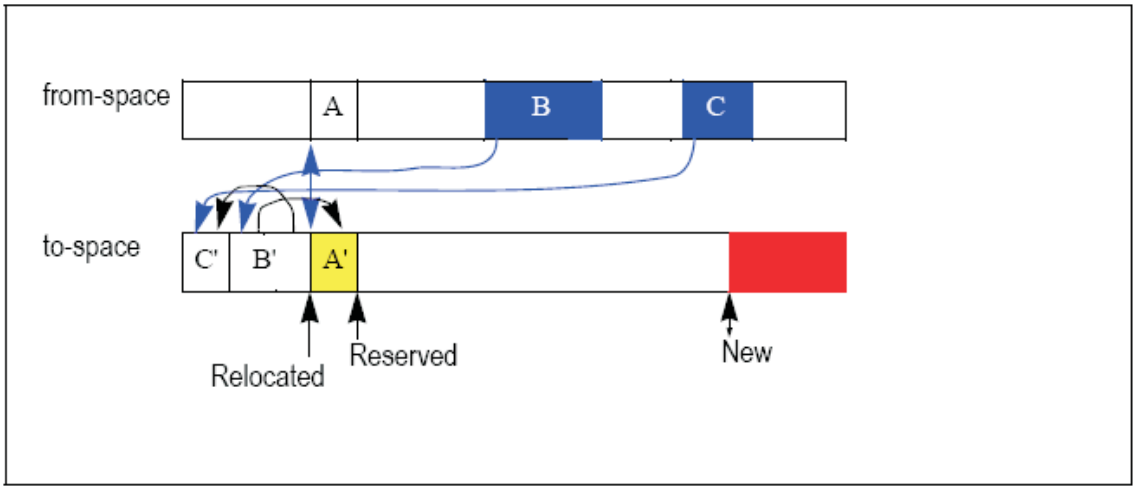

Figure 1. Incremental Copying Garbage Collection

this object has not yet been relocated. Each of the invalid versions maintains a pointer to the valid version, colored blue in the illustration. Objects waiting to be relocated, such as A, also maintain a forwarding pointer to the memory that has been reserved to hold the eventual copy. As objects are being relocated, each pointer contained within the object is replaced by a pointer to the new to-space copy of the referenced object. Thus, object B' holds pointers to A' and C', whereas object B held pointers to A and C. A beneficial side effect of copying garbage collection is that the unused memory scattered throughout from-space is coalesced into a single larger free segment, from which new memory requests that are issued while garbage collection is taking place can be served.

The PERC virtual machine divides the memory allocation pool into multiple equal-sized regions. On each pass of the garbage collector, two regions are selected as to- and from-space respectively. These regions are defragmented using the incremental copying garbage collection technique. The unused memory in the other regions is reclaimed using an incremental mark and sweep technique which does not relocate objects. In typical situations, the mark-and-sweep technique achieves the highest memory utilization, but runs the risk of arbitrarily poor utilization in the rare event that it experiences severe memory fragmentation. Incremental copying garbage collection achieves guaranteed utilization of approximately 50%. Depending on workload characteristics and risk-reward profiles, users of the PERC virtual machine can configure the memory allocation pool for a small number of very large regions (with guaranteed defragmentation, but lower expected memory utilization) or a large number of relatively smaller regions (with better typical memory utilization, but higher risk of fragmentation degradation).

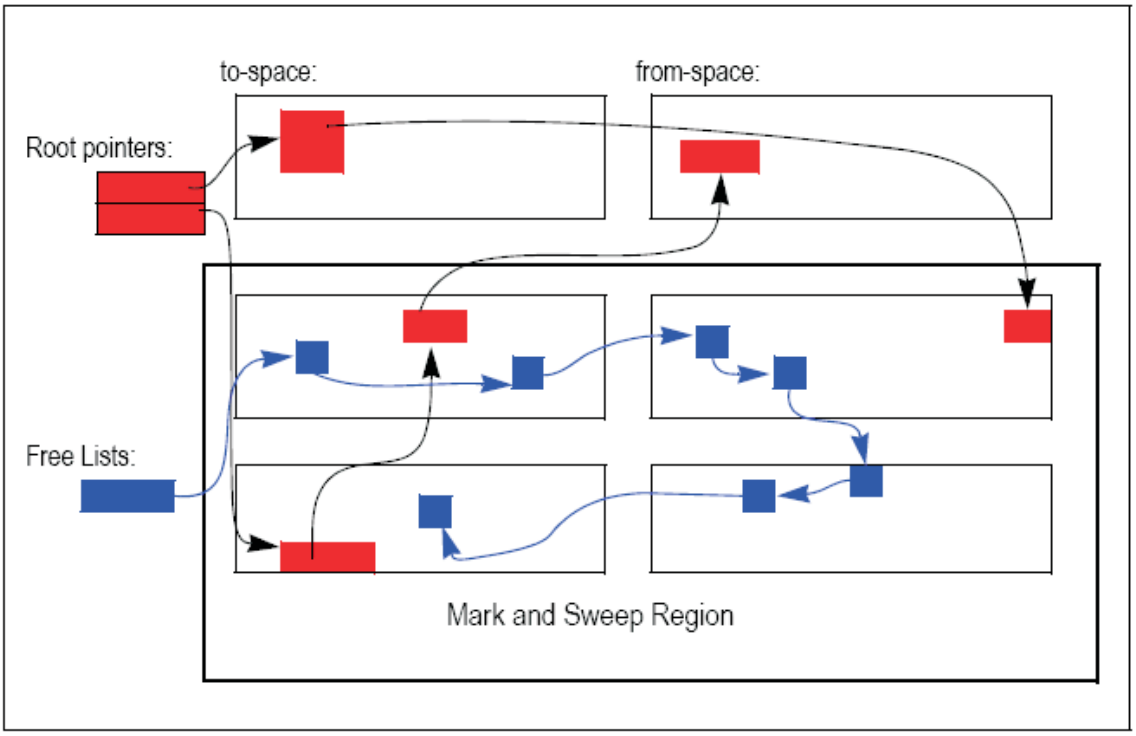

Figure 2. Mostly Stationary Real-Time Garbage Collection

An important unique attribute of the PERC garbage collector is that the total effort required to complete garbage collection is bounded by a configuration-dependent constant, regardless of how much memory has recently been allocated or discarded, and independent of how many times the garbage collector is preempted by application threads. Given this property, it is straightforward to schedule garbage collection to periodically reclaim all of the dead memory in the system. The VM Management API allows garbage collection scheduling parameters to be adjusted on the fly in order to accomodate changes in the system workload. We call this garbage collection pacing. This makes sure that the system never exhausts its allocation free pool. If a virtual machine does not support pacing of garbage collection, then it is possible to experience situations in which a low-priority task allocates memory subsequently desired by a high priority task, thereby forcing the high priority task to wait for garbage collection to complete before it can advance. This is an example of priority inversion that is nearly impossible to avoid with most implementations of the Java virtual machine, but is easily avoided with the PERC virtual machine.

The objective of garbage collection pacing is to make sure that garbage collection gets enough increments of CPU time to make sure it consistently replenishes the allocation pool before the available supply of memory has been exhausted. Figure 3 shows a simulated air traffic control workload with real-time garbage collection running under the direction of a real-time pacing agent.

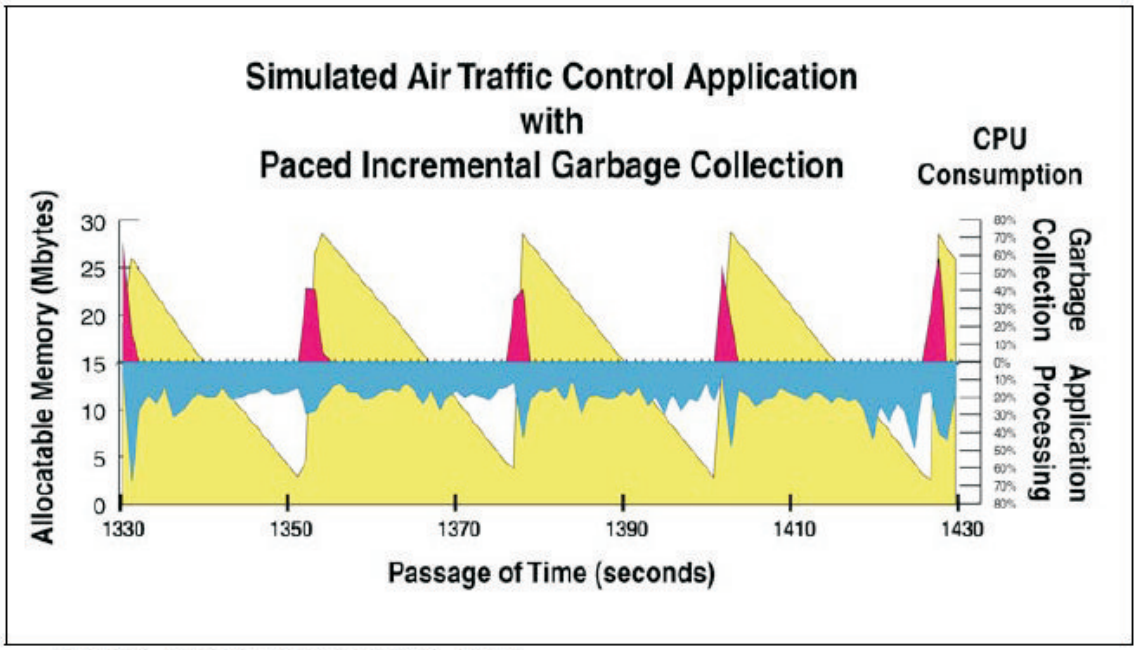

Figure 3. Allocatable Memory vs. Time

This particular application is running in a fairly predictable steady state as characterized by the following observations. First, the slope of the yellow chart, which represents the amount of memory available for allocation, is roughly constant whenever garbage collection is idle. This means the application's allocation rate is approximately constant. Second, the heights of the yellow chart's peaks are roughly identical. This means the amount of live memory retained by the application is roughly constant. In other words, the application is allocating new objects at approximately the same rate it is discarding old objects. Finally, the percentage of CPU time required for application processing is well behaved, ranging from about 20% to 50%.

Note that garbage collection is idle most of the time. As memory becomes more scarce, garbage collection begins to run. When garbage collection runs, it interferes with some, but not all, of the real-time application processing. For this reason, you'll note a slight dip in application processing each time the garbage collector, represented by the occa-

sional red upward spikes, runs. You'll also note a tendency for application processing to briefly increase following each burst of garbage collection. This is because the pre-empted application needs to perform a small amount of extra work to catch up with real-time scheduling constraints following each preemption.

When properly configured, the pacing agent will carefully avoid delaying the application threads by any more than the allowed scheduling jitter. Configuration parameters to the pacing agent allow specification of the priorities at which garbage collection runs, safety buffers to accommodate transient bursts of high workload activity, and a maximum CPU utilization budget for garbage collection activities. This allows the garbage collection activities to be treated the same as application taks in the rate monotonic analysis of schedulability.

PRIORITY INHERITANCE

Because Java was designed to support multiple threads, the language has built-in support for synchronization locks. A synchronization lock allows an individual thread to lock out other threads from performing certain operations while the first thread manipulates shared data. To avoid priority inversion, in which a medium priority thread indirectly prevents a high priority thread from making forward progress, the PERC virtual machine supports a protocol known as priority inheritance. When the high-priority task blocks because it needs access to a lock held by a lower-priority thread, the low-priority thread inherits the high priority of the requesting thread. This allows the low priority thread to temporarily elevate its priority in order to preempt any intermediate-priority threads so that the low-priority thread can finish its work and get out of the way of the high-priority thread that desires access to the shared lock.

PRECISE TIMERS

Another example of a way in which the PERC virtual machine has been customized to better support the needs of real-time programmers is our real-time implementation of the standard java.util.Timer class. This class provides an ability to schedule periodic execution of certain Java tasks. The problem with the traditional implementation of these services is that if someone adjusts the real-time clock of the underlying operating system, the periodic behavior of tasks is compromised. Prompted by a key telecommunications customer, Atego developed an alternative to java.util.Timer in order to provide consistent periodic execution of timer tasks, even if the system clock is modified during execution.

SYMMETRIC MULTI-PROCESSOR (SMP) SUPPORT

The PERC Ultra virtual machine has recently been enhanced to support real-time operation also on multiprocessor systems. PERC Ultra SMP will fully exploit all of the processors available on your underlying SMP architecture as long as your application has been structured so that sufficient numbers of Java threads are always ready to run. The garbage collection algorithms have been enhanced so that the total effort of garbage collection can be divided between multiple processors. Thus, the wall-clock time required to complete garbage collection on a 4-processor platform is approximately one fourth the time required to collect the same total amount of garbage on a single processor. Pacing of garbage collection in the SMP environment uses the same techniques that are used in the uniprocessor version of PERC Ultra.

Platform Support

One of the special challenges associated with supporting embedded Java development is the large number of processors, real-time operating systems, and board support packages. Atego has made significant investments in porting, testing, and maintaining the PERC software on a breadth of platforms. Many of these platforms are tested around the clock in Atego' QA laboratory. The PERC VM has been deployed on many different platforms, including:

1. **Operating Systems and Real-Time Operating Systems**

- Green Hills INTEGRITY
- LynuxWorks LynxOS, LynxOS-178 and LynxOS-SE
- Microsoft Windows NT, Windows Mobile
- ENEA OSE
- QNX Neutrino
- Sysgo PikeOS
- Wind River VxWorks, VxWorks MILS, and VxSim

2. **Linux Operating System Distributions**

- MontaVista Linux Professional Edition, Carrier Grade Linux
- Denx ELDK
- Fedora Core
- Freescale Linx
- Red Hat Enterprise Linux
- RedHawk Linux
- SUSE
- Sysgo ELinOS
- Wind River Linux
- YellowDog Linux

3. **Target Processors**

- Interpreted execution for 68000, MIPS, SH4
- JIT and AOT compilers and interpreted execution for Pentium, PowerPC, XScale, ARM, and StrongARM

Company Responsiveness

CUSTOMIZATION

Atego has always focused its efforts on the special needs of the embedded developer. Over the years, we have enhanced PERC products with a variety of capabilities developed in response to specific requests from particular customers. Generally, the enhancements requested by particular customers are added to the core products to benefit all customers. Some examples of the custom features we have developed include:

- Symbolic debugging of AOT-compiled code. An earlier implementation of the PERC virtual machine supported symbolic debugging only of interpreted code. A key telecommunications customer told us they needed symbolic debugging of AOT-compiled code as well. Today, we have the only embedded VM on the market supporting this feature.

- Fail-over RMI implementation. The standard remote method invocation (RMI) libraries defined by Sun do not allow hot-swappable replacements of RMI servers. At the request of a large telecommunications equipment provider, we modified the RMI implementation to allow hot-swapped replacement cards to take over the remote method services of the cards they are replacing.

- Direct memory implementation. A startup telecommunications equipment vendor sought a mechanism to achieve more efficient sharing of information between native C and Java components. We worked with this customer to define a direct memory class which provides native methods for accessing native memory objects. Then we taught our JIT and AOT compilers to treat this class specially, in-lining the operations to eliminate the overhead of JNI calls.

- Precise Timers. The java.util.Timer class provided by Sun schedules periodic activities. The standard implementation exhibits significant shifting of the period whenever the system clock is altered. Atego worked with one of its real-time customers to develop a periodic timer that behaves consistently, independent of any changes to the system clock.

- TFTP Classloader and Target Server. We have found that many embedded computers lack file systems. For one such system, we worked with an industrial automation customer to define and implement a TFTP class loader and corresponding target server (to provide the classes that need to be loaded). This class loader supports traditional class files in addition to zip and jar files.
- Improved Debugging Support. A customer found it desirable to view source code line numbers in stack backtraces associated with native-compiled code (both JIT and AOT). Atego modified the virtual machine implementation to offer this capability, even for the deployment version (as opposed to the development/debugging version) of the virtual machine.

DEVELOPER SUPPORT

Much more than enterprise developers, embedded developers face a bewildering assortment of interoperability challenges and configuration choices. There is no such thing as an "industry standard embedded platform". Availability of high-quality technical support is critical to the success of your project.

Atego developer support leads the industry. Existing customers are quick to praise our competence and willingness to assist.

PROFESSIONAL SERVICES

Atego maintains a dedicated staff of professional engineers who are available to assist with training, architecture and design consulting, and in some cases, development of complete applications.

Summary

The unique features of the PERC virtual machine make it ideal for development of management-plane software for network element applications such as optical platforms, multi-service access devices, service concentration equipment, and IP infrastructure; control and management software for distributed industrial automation applications; and communication software for fleet telematics. Adopting PERC today allows you to more rapidly add greater functionality and differentiation to the software in your deeply embedded products.

Atego's PERC virtual machine is the only embedded Java virtual machine to deliver the full generality of J2SE libraries; the performance benefits of static linking and native compilation; the reliability and predictability benefits of unique patent-protected real-time garbage collection; and developer tool support for symbolic debugging, performance profiling, and interactive VM management.

To obtain more information, please contact Atego at **www.atego.com** or call one of our sales offices

North America
Phone: (888) 91-ATEGO
Fax: (858) 824-0212
E-mail: info@atego.com

France
Phone: +33 (0) 1 4146-1999
Fax: +33 (0) 1 4146-1990
E-mail: info@atego.com

United Kingdom
Phone: +44 (0) 1491 415000
Fax: +44 (0) 1491 575033
E-mail: info@atego.com

Germany
Phone: +49 7243 5318-0
Fax: +49 7243 5318-78
E-mail: info@atego.com

© 2010 Atego. All rights reserved. Atego™ is a trademark of Atego. PERC® is a registered trademarks or service mark of Atego. Java™ and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc., in the US and other countries. All other company and product names are the trademarks of their respective companies.