Aries: Transparent Execution of PA-RISC Applications on IA-64

Vania Ning Fang

Mike (Xiaoxin) Chen

Hewlett-Packard Company
11000 Wolfe Road, MS42U5
Cupertino, CA

Tel: (408) 447-0965
Fax: (408) 447-4924

{vania, xchen}@cup.hp.com

Introduction

Over the course of years, new computer architectures have been designed and developed to address known issues of their predecessors or to accommodate the increasing performance need required by the computing community. As new architectures emerge, however, users inevitably face the problem of having to migrate applications built on the previous platform(s) to the new platform(s). The migration process usually requires a considerable amount of user intervention and is typically very time consuming. Having to repeat the lengthy porting process for each application incurs extra overhead on users. An alternate solution to this problem is dynamic binary translation - a technology that transparently translates binaries from one instruction set to the other and executes the translated code immediately after the translation becomes available. Applications on the old platform can be executed in the same manner on the new platform without major performance penalty or extensive maintenance cost.

In order to ensure smooth transition from Hewlett-Packard's PA-RISC[1] architecture to Intel's IA-64 architecture, Hewlett-Packard has developed Aries - a software emulator that accurately translates PA-RISC binaries to native IA-64 code using the dynamic binary translation technology. As the only software migration engine available for PA-RISC to IA-64 translation, Aries innovates the traditional software emulators in that it offers not only transparency between different computing platforms, but also excellent reliability and sustained performance. Unlike previous emulators or translators, Aries is designed to cover all PA-RISC applications and preserve the necessary performance requirements for the applications.

Motivation

Porting applications between platforms, especially when the underlying computer architecture changes, typically involves modification and recompilation of the application source or binary. This traditional approach evidentially incurs a lot of overhead that can be otherwise prevented by adopting the dynamic translation technology. First of all, availability of application source code is limited - without source code, recompilation is simply out of the question, as is the case for legacy applications. Secondly, given the source code, recompilation of applications is still not an easy task. Users have to rely on the application designer to manually make changes to the source code in order to adapt to the new architecture platform, which is cumbersome and time consuming. In addition, some changes cannot be performed without the aid of certain porting tools, which again are not always readily available. Thirdly, a majority of the applications have more than one version as they evolve over years. For each individual version users wish to execute on the new platform, porting has to be done separately. In addition, certain versions might require additional effort to port due to version specific implementation details.

Binary translation, on the other hand, offers smooth transition for all applications from one architecture to the other. It requires no user intervention, nor does it depend on availability of source code or any porting tools. From user's perspective, applications can be executed on the new platform in the same manner as on the old platform. Dealing with various versions of an application is also not an issue, since each version will simply be translated and executed separately as if they are different applications. Binary translation offers transparency that cannot be achieved via traditional porting approach. As such, a state-of-art binary emulation engine like Aries is a natural solution to the software migration problem.

Besides transparency, completeness is another important feature of Aries that differentiates it from a mere research prototype. Completeness means that Aries is capable of emulating all PA-RISC applications. This requires not only machine instruction emulation, but also accurate simulation of the old runtime environment based on the new one. In addition, Aries maintains application performance comparable to the PA-RISC performance. All of the above

[1] PA-RISC stands for Precision-Architecture-reduced-instruction-set-computing

ensure that users can easily execute their PA-RISC applications on the IA-64 platform without noticeable difference.

High Level Overview

Aries is a software emulator that is designed to meet the following requirements without introducing any security holes for the applications it emulates:

1. Hardware level reliability
2. Transparency
3. Comparable performance to native PA-RISC execution

On all IA-64 machines bundled with Aries, users can install their PA-RISC applications and launch them just as they would on PA-RISC systems. The HP-UX kernel on the IA-64 machine will detect that the application is not a native IA-64 binary; the control is then transferred to Aries from this point on. Aries will faithfully emulate the PA-RISC application without requiring any effort on user's side.

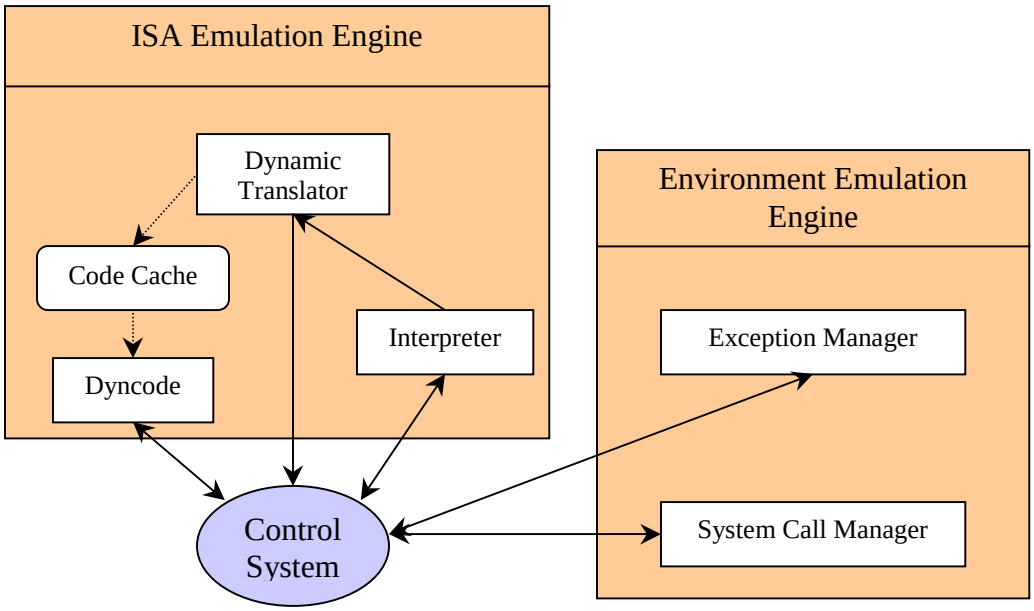

Figure 1. Aries Emulation System

Data Flow
Control Flow

Aries emulates a PA-RISC application by emulating the program's instructions, the program's system calls, and the behavior of the HP-UX/PA-RISC kernel. Figure 1 shows the control flow among Aries components. The core of Aries is the control system, which is responsible for dispatching between components of the emulation engine. Aries' emulation engine can be categorized into two modules - namely the Instruction Set Architecture (ISA) emulation engine and the operating system environment emulation engine. The ISA emulation module deploys a combination of fast interpretation and dynamic translation technology to ensure reliable and efficient instruction emulation. Each PA-RISC block is by default emulated by the fast interpreter until it hits the translation threshold, by which time it will be sent to the dynamic translator. The dynamic translator translates the PA-RISC block to the equivalent set of IA-64 instructions. Such translated executable code is referred to as Dyncode and is stored in Aries code cache for subsequent use. The next time the same PA-RISC block is encountered, Aries will directly execute the corresponding Dyncode without incurring additional translation overhead. The environment emulation module is responsible for processing system service requests made

by the PA-RISC applications as well as signals delivered to the applications by the HP-UX operating system. More details about each of these components will be discussed in the following section.

The Underlying Technology

1. Control System

Aries can be thought of as a state machine -- the control moves from one state to the other during the emulation of a PA-RISC application. The control system is responsible for all of the state transitions.

A PA-RISC application can be considered as a set of sequential PA-RISC code blocks and the program control jumps from one code block to the other. Aries emulates one block at a time and transition to other code blocks occurs when the emulation of the current block completes. In Aries, a PA-RISC code block can be either interpreted or emulated by executing the native IA-64 code, which is previously translated by the dynamic translator at run time. When the number of times a PA-RISC block has been interpreted reaches a threshold, it is identified as a "hot block". A "hot block" is immediately translated into native IA-64 code (here after referred to as Dyncode) by the dynamic translator and then stored in the Aries code cache. From now on, Aries control system will directly branch to the corresponding Dyncode for each subsequent invocation of the same PA-RISC block.

An application makes service requests to the underlying system via system calls, the interface between the application and the operating system. Aries intercepts all of the system calls made by the PA-RISC application and maps them to the corresponding system call stub routines, which can either be direct IA-64 system calls or Aries emulated routines. Since Aries emulates PA-RISC applications on the HP-UX operating system, most of the system calls are passed through to the equivalent IA-64 system calls. However, it is incorrect to invoke the native IA-64 routine for some special system calls, such as system calls related to the runtime context and PA-RISC machine status. These system calls are implemented by Aries.

On a PA-RISC system, when the operating system delivers a signal to the PA-RISC application, the application can choose to block or pass it to the corresponding signal handler for this signal. The signal handler interrupts the normal program execution and returns control back to the program after processing the signal. When Aries emulates a PA-RISC application, it has to preserve the same signal behavior. All signals are intercepted and recorded by Aries and the control system then delivers to the user specified signal handler at the appropriate time.

2. ISA Emulation

a. Fast Interpreter

The Aries interpreter fetches and decodes PA-RISC instructions one at a time and executes it based on Aries maintained PA-RISC machine states, which contains all PA-RISC registers and other program status fields. The interpreter is carefully written in C for optimal performance. It is capable of interpreting all PA-RISC code blocks including the ones that are too complex to be translated by the dynamic translator. The interpreter transfers control back to the control system after executing one PA-RISC code block. Figure 2 shows the interaction between the fast interpreter and the dynamic translator.

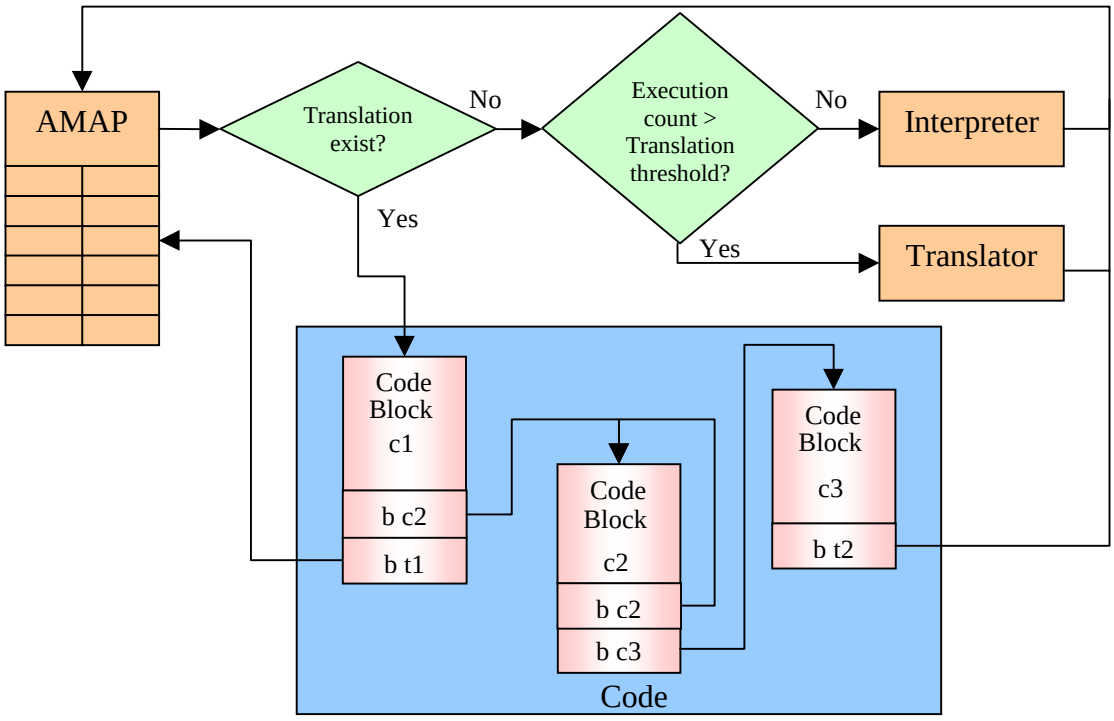

Figure 2. Aries interpretation and dynamic translation

b. Dynamic Translator

When invoked by the Aries control system, the dynamic translator translates a PA-RISC code block into a Dyncode block, which is a set of equivalent native IA-64 instructions. It then stores the Dyncode in Aries code cache so that the control system can directly jump to it when the same PA-RISC block is invoked again. Aries maintains an Address Lookup Table (AMAP) that maps from PA-RISC code blocks to the corresponding Dyncode. For each PA-RISC block, the control system will first attempt to look it up in the AMAP. If translation for this PA-RISC code block already exists, the control system will directly execute the Dyncode. Otherwise, the control system will either interpret or translate the block depending on the translation threshold.

The translator consists of the decoder, code generator and the instruction scheduler. The decoder simply parses each PA-RISC instruction and converts it to internal representation to be handled by the code generator. The code generator translates each PA-RISC instruction into a sequence of equivalent native IA-64 instructions. All 128 IA-64 general registers are used in the translated code. To boost performance, all PA-RISC general registers are mapped to IA-64 registers so that the translated IA-64 code will not make more memory references than the original PA-RISC code block. The rest of the IA-64 registers are used to store intermediate values to maximize instruction level parallelism.

IA-64 architecture uses instruction bundle to hold up to three instructions in predefined format. In addition, instructions might be executed out of order on IA-64 while PA-RISC programs are executed in strict instruction order. These incur extra difficulties in Aries implementation because Aries needs to encode the translated IA-64 instructions in bundles and explicitly put stop bits to preserve program order if necessary. Aries scheduler efficiently schedules instructions into bundles and insert stop bits in an optimal way to ensure program correctness.

c. Dynloop

To speed up execution of Dyncode blocks, an assembly routine (Dynloop) was implemented to transition from one code block to the other. Dynloop maintains a direct lookup table that maps

a PA-RISC code block address to the matching Dyncode block. If the lookup succeeds, it directly jumps to the target Dyncode block. Otherwise, it returns back to the control system, which will perform a more expensive AMAP lookup.

More over, Aries implements a backpatch technique that allows a Dyncode block to directly branch to another Dyncode block without going through a target lookup. When the Dyncode block is first translated, the branch targets are unknown because they do not statically map to a Dyncode block. When the Dyncode block corresponding to the PA-RISC branch target becomes available, Aries modifies the branch instruction in the previous Dyncode block so that it jumps to the target Dyncode block.

3. Environment Emulation

a. System Call Manager

All PA-RISC HP-UX system calls enter the kernel space through a common system call gateway page. The environment emulation module captures system calls made in an emulated PA-RISC application at the gateway page and calls the corresponding emulation routines. Most system call emulation routines are simple stubs that invoke the native system calls directly on the IA-64 platform. Other system calls require special handling. For example, when the PA-RISC application sets the signal mask. Aries intercepts this system call because Aries maintains the application's signal mask.

b. Exception Manager

Signals can be raised synchronously or asynchronously. Synchronous signals are always delivered at the point of the faulting instruction while asynchronous signals do not have a fixed point of delivery. In other words, it is possible for a program to receive an asynchronous signal at different points of execution in separate runs. Aries simulates the exact behavior.

All signals raised by the operating system are captured and recorded by Aries. Each synchronous signal is delivered immediately after emulating the faulting instruction. For asynchronous signals, Aries queues them up and deliver them to the emulated application at the earliest locations where it can construct a correct PA-RISC signal context.

A slight complexity arises when a signal occurs during execution of a Dyncode block. If the 1signal is synchronous, Aries commits all instructions prior to the faulting instruction and delivers the signal to the user specified handler immediately. If it is an asynchronous signal, Aries delays the delivery until the current Dyncode block has finished.

Special care should also be taken when signals arrive in the middle of a system call. On PA-RISC systems, the system call will either be aborted or completed, and the signal is delivered upon the system call return. Aries decides whether to abort or complete the system call depending on the point when the signal arrives. If the signal arrives before the system call return point, the system call is aborted. Otherwise, the system call returns with the status provided by the native IA-64 system call.

Verification Methods

To achieve hardware level reliability has always been Aries' top priority goal. Given the functionality of Aries, application testing is a necessary step toward achieving the quality goal. Aries has been successfully tested with applications of various flavors on the IA-64 platform, including signal intensive applications, user interactive applications and other applications of different durations. Nevertheless, application testing is not sufficient to cover all of Aries source code base. As such, the Aries team has come up with several innovative verification methods, which have effectively increased Aries reliability.

A random testing framework was developed to stress test Aries' ISA emulation engine. The random test engine consists of a server and a thin client. The server randomly generates a set of PA-RISC instruction sequences. The generated PA-RISC code sequence is then executed by the server on a PA-RISC machine and by the client on an IA-64 machine under Aries. The final program states are compared for each PA-RISC code block and any discrepancies between the PA-RISC execution and IA-64 execution indicate an Aries failure. This random test engine ensures excellent coverage of Aries ISA emulation and it has been instrumental in improving Aries' ISA emulation quality.

The Aries team also developed a cross validation tool that automatically verifies Aries at run time. This tool enables Aries engineers to pinpoint failures at the exact PA-RISC code block where the bug is first originated, which is extremely helpful since many bugs are not manifested until much later in the execution sequence, making it difficult to track down the source of failure.

Performance Statistics

Aries performance is measured on a 500MHZ Itanium box. The benchmark applications used in these measurements include SpecInt95 and SpecFP95. Figure 3 shows Aries IA-64 performance of integer applications relative to a K-class PA-RISC machine. Figure 4 shows Aries IA-64 performance of floating point applications relative to a K-class PA-RISC machine. The overall Aries IA-64 integer performance is approximately 3.0 times as slow as the native PA-RISC execution. Floating point applications are about 3.2 times as slow as the PA-RISC performance.

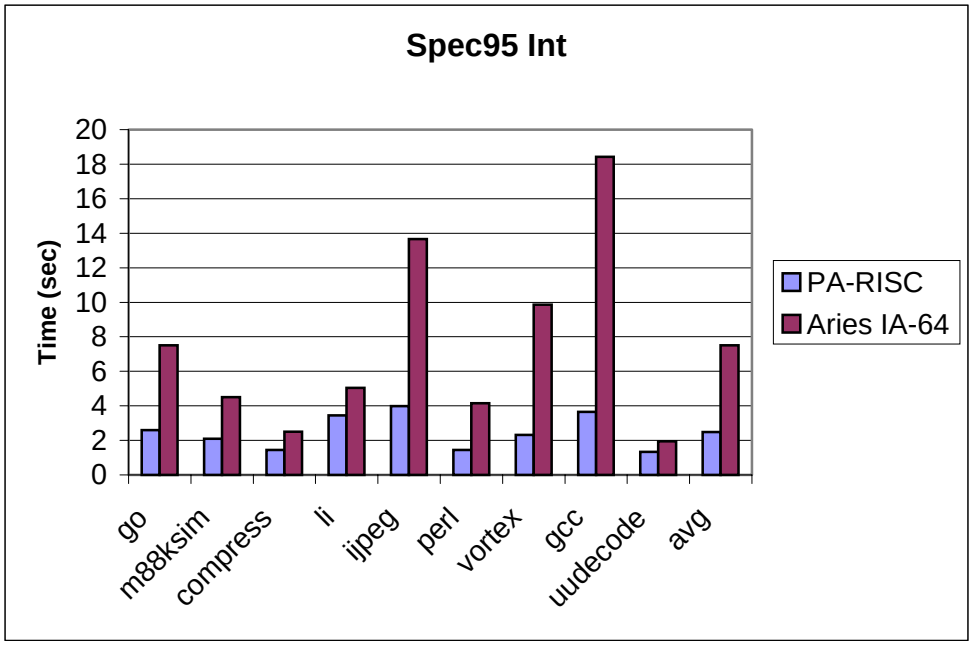

Figure 3. Aries Performance for Spec95 Integer Benchmarks[2]

[2] Data collected based on May 29th version of Aries

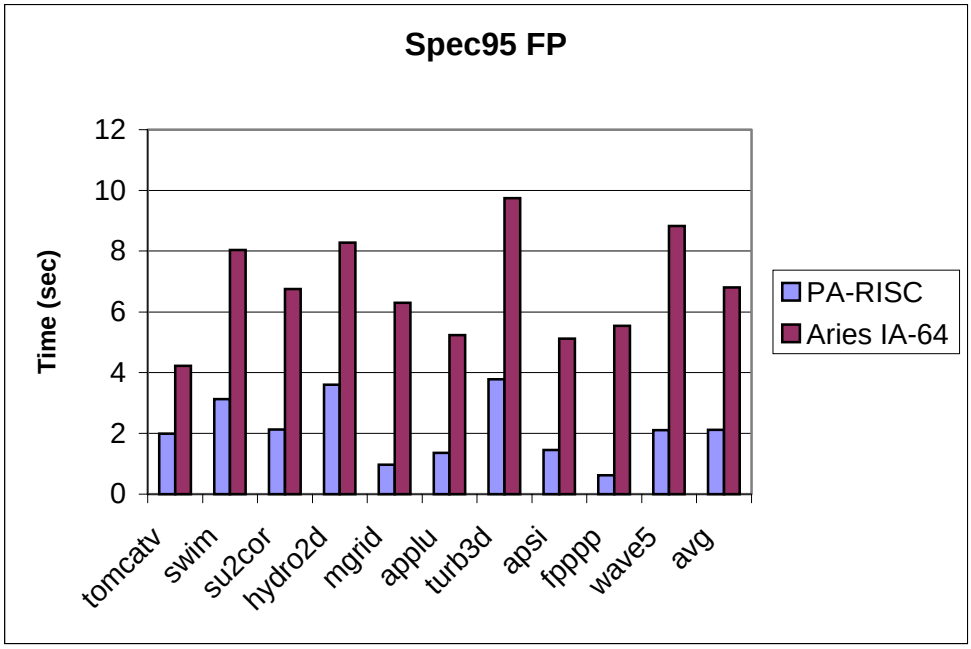

Figure 4. Aries Performance for Spec95 Floating Point Benchmarks

Conclusion

As the only PA-RISC to IA-64 software emulator available, Aries transparently and effectively emulates all user-level applications built for HP-UX/PA-RISC systems. Aries deploys a combination of fast interpretation and dynamic translation technology to emulate the PA-RISC instruction set architecture efficiently. Aries also faithfully simulates the HP-UX/PA-RISC system behavior to emulate all PA-RISC applications accurately. In addition, Aries keeps up a performance level comparable to that of native PA-RISC systems.

References:

[1] C. Zheng, C. Thompson, PA-RISC to IA-64: Transparent Execution, No Recompilation. *Compute*, Vol. 33, No. 3, pages 47-52, March 2000.