Aries: Transparent Execution of PA-RISC/HP-UX Applications on IPF/HP-UX

Keerthi Bhushan

Rajesh K Chaurasia

Hewlett-Packard
India Software Operations
29, Cunningham Road
Bangalore – 560 052
India

+91-80-2251554
(408) 447-3386

keerthi@cup.hp.com
rajeshkc@india.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). This 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/HP-UX[1] platform to Intel's IPF/HP-UX platform, Hewlett-Packard has developed Aries - a software emulator that accurately translates PA-RISC/HP-UX (both 32-bit and 64-bit)[2] binaries to native IPF/HP-UX code using the dynamic binary translation technology. Aries offers not only transparency between the two computing platforms, but also excellent reliability, correctness, completeness and sustained performance[3].

Motivation

Binary translation offers the following solutions in the area of migration between computing platforms:

1. Eliminates the overhead of recompilation of application source, with added time and cost of maintaining the ported application
2. Provides migration path for applications whose source codes are not available for porting
3. Provides migration path for applications which cannot be ported without aid of porting tools, which may not be readily available
4. Eliminates the cost of porting multiple versions of applications, as the evolve, and maintaining them

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/HP-UX 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 acceptable levels of performance vis-à-vis the native PA-RISC/HP-UX system performance. All of the above ensure that users can easily execute their PA-RISC/HP-UX applications on the IPF/HP-UX platform with correctness assured.

---

[1] PA-RISC stands for Precision-Architecture-reduced-instruction-set-computing. PA1.x is a 32-bit architecture, and PA2.0 is a 64-bit architecture.

[2] Aries32 and Aries64 are both available for 32-bit and 64-bit PA-RISC applications, respectively.

[3] See Aries performance report at the end of this document.

High Level Overview

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

1. Very high level of reliability
2. Transparency
3. Acceptable performance vis-à-vis native PA-RISC execution

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

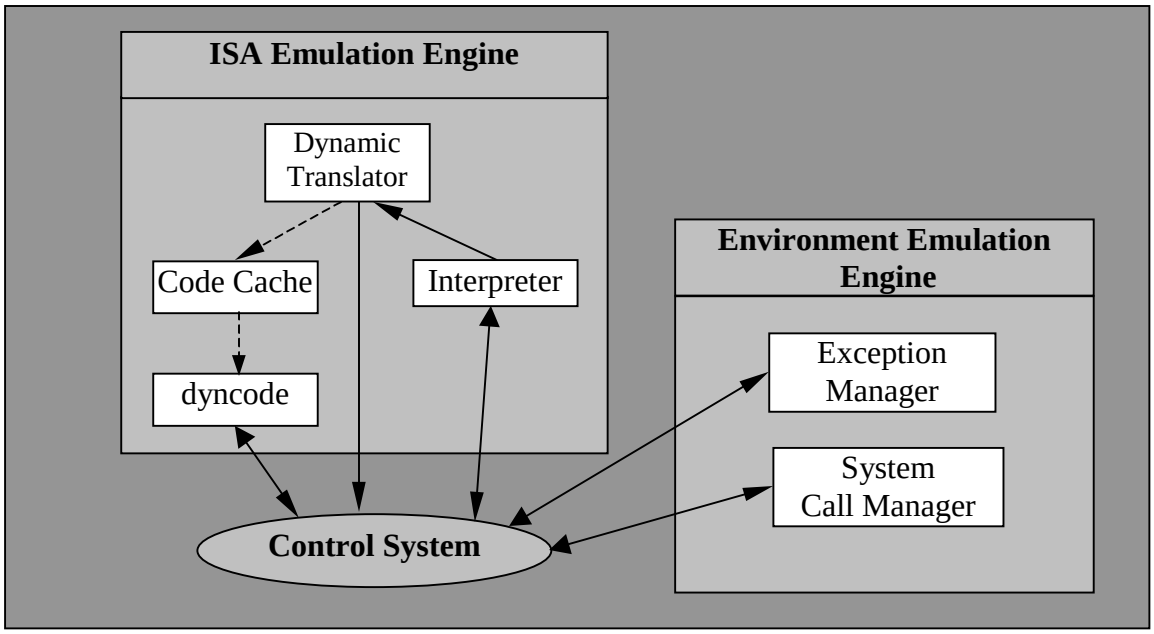

Figure 1. Aries Emulation System

Data Flow

Control Flow

Aries emulates a PA-RISC/HP-UX application by emulating the program's instructions, the program's system calls, and the behavior of the PA-RISC/HP-UX kernel. Figure 1 shows the interaction between various Aries' components.

The core of Aries is the Control System, which is responsible for dispatching data and control between various components of the emulation engine. Aries emulation engine can be categorized into two modules, viz., the Instruction Set Architecture (ISA) emulation engine and the operating system Environment Emulation Engine (EEE).

The ISA emulation engine deploys a combination of fast interpretation and dynamic translation technology to ensure reliable and efficient instruction emulation. Each PA-RISC code block[4] 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 code block to the equivalent set of IPF instructions. Such translated executable code is referred to as

[4] A code block is a set of machine instructions without a control flow break in between.

dyncode and is stored in Aries *code cache* for subsequent use. Next time when the same PA-RISC block is encountered, Aries will jump to the corresponding dyncode block, which executes at native IPF speed without incurring additional overhead.

The environment emulation module is responsible for processing system service requests made by the PA-RISC/HP-UX application as well as correctly relaying signals delivered by the HP-UX operating system to the emulated application. More details about each of these components will be discussed in the following section.

The Underlying Technology

1. Control System

The Control System maintains the emulated PA application context and also acts as a switch between the various components in ISA and EEE.

Aries considers the emulated application as a set of PA-RISC machine code blocks, with inter-block branches. The emulation starts with the interpretation of the first of such code blocks. The interpretation proceeds on a block-by-block basis. After each block is interpreted, the Control System takes control and checks for the next block to be emulated. It checks if translation exists for the next block. If yes, the control jumps to the translated code block, which executes at the speed of the native IPF platform. If not, then the Control System may choose the code block for translation. This choice is based upon the count that the Control System keeps of how many times the concerned code block has been interpreted till then. If this count has reached the translation threshold, the code block is dispatched to the translator for translation. The PA-RISC machine code blocks that have hit the translation threshold are called *hot blocks*. *Hot blocks* represent the mostly executed set of code blocks on the emulated application. Translating only the *hot blocks* speeds up the emulation since we will then always find the most required blocks in the *code cache.*

An application makes service requests to the underlying operating 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/HP-UX application and maps them to the corresponding system call stub routines, which can either be direct IPF/HP-UX system calls or Aries emulated routines. Since Aries emulates PA-RISC/HP-UX applications on the IPF/HP-UX operating system, most (90%, actually) of the system calls are passed through to the equivalent IPF/HP-UX system calls. However, it is incorrect to invoke the native IPF/HP-UX routine for some special system calls, such as system calls related to the runtime context and PA-RISC machine status. Aries implements these system calls.

On a PA-RISC/HP-UX system, when the operating system delivers a signal to the PA-RISC/HP-UX application, the application can choose to,

- Take default action
- Block or ignore the signal
- Handle the signal with a signal handler

When Aries emulates a PA-RISC/HP-UX application, it preserves exactly the same signal behavior. All signals are intercepted and recorded by Aries and the Control System then delivers them to the emulated application at the appropriate time.

2. ISA Emulation

a. Fast Interpreter

The Aries interpreter fetches and decodes PA-RISC instructions from a code block one at a time and executes it based on Aries-maintained PA-RISC machine context (state), which

contains all PA-RISC registers and other program status fields. The interpreter is carefully written in C for optimal performance. 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 via the Control System.

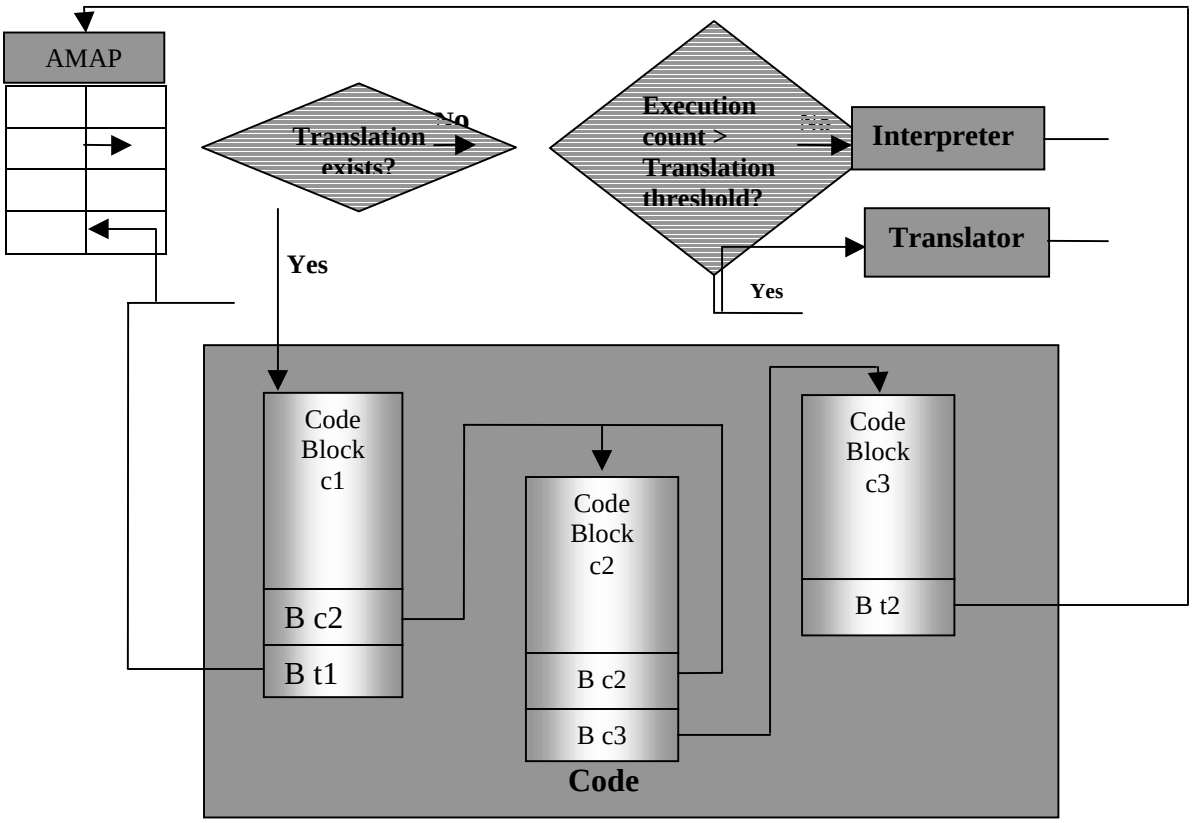

Figure 2. Aries interpretation and dynamic translation

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 IPF machine instructions. 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 the 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 IPF instructions. All 128 IPF general registers are used in the translated code. To boost performance, all PA-RISC general registers are mapped to IPF registers so that the translated IPF code will not make more memory references than the original PA-RISC code block. The rest of the IPF registers are used to store intermediate values to maximize instruction level parallelism.

IPF architecture uses instruction bundles to hold up to three instructions in a predefined

format. Aries scheduler efficiently schedules instructions into bundles and insert stop bits in an optimal way to ensure program order and correctness.

c. dynloop

To speed up execution of *dyncode* blocks, an assembly language 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.

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/HP-UX 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 IPF/HP-UX platform. Other system calls require special handling. For example, when the PA-RISC/HP-UX 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 this 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/HP-UX signal context. A slight complexity arises when a signal occurs during execution of a dyncode block. If the signal 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/HP-UX 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 IPF/HP-UX system call.

Verification Methods

To achieve hardware levels of reliability has been Aries' top priority. Given the functionality of Aries, application testing is a necessary step toward achieving this goal. Aries has

been successfully tested with various types of applications-user-interactive applications and compute-intensive applications. The following table lists some of the tested applications.

Compute-intensive applications	System-intensive/User-interactive applications
NASTRAN	Netscape Browser
JAVA Virtual Machine	XEmacs
Spec'2000 and Spec'95	SAM
Perl	Apache Web Server
Alaska	Real Server
gcc	Zeus Web Server

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 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/HP-UX machine and by the client on an IPF/HP-UX machine under Aries. The final program states are compared for each PA-RISC code block and any discrepancies between the PA-RISC/HP-UX execution and IPF/HP-UX 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. Millions of PA-RISC instruction sequences have been run without a single failure, which ensures good test coverage of all instructions.

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 733MHz Itanium box. The benchmark applications used in these measurements include SpecInt2000 and SpecFP2000.

Figure 3 shows Aries' IPF/HP-UX performance of integer applications relative to a PA-RISC machine with PA8600 processor at 550MHz.

Figure 4 shows Aries' IPF/HP-UX performance of floating point applications relative to a PA-RISC machine with PA8600 processor at 550 MHz.

Integer applications show a slowdown factor of 2X to 4x. Floating-point applications show a slowdown factor of 3X to 6x.

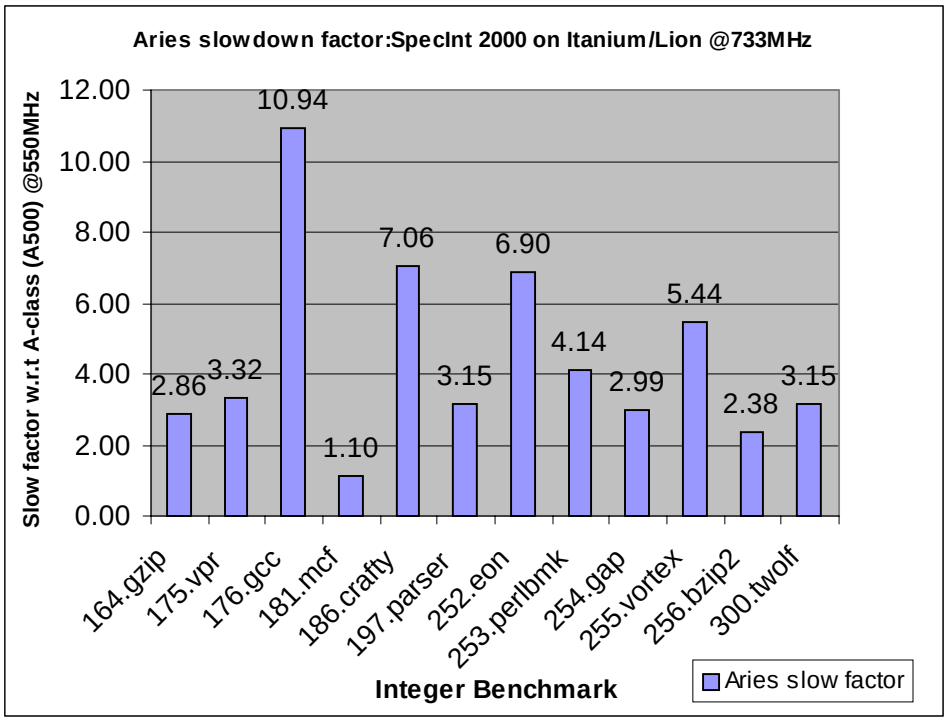

Figure 3. Aries Performance for Spec2000 Integer Benchmarks

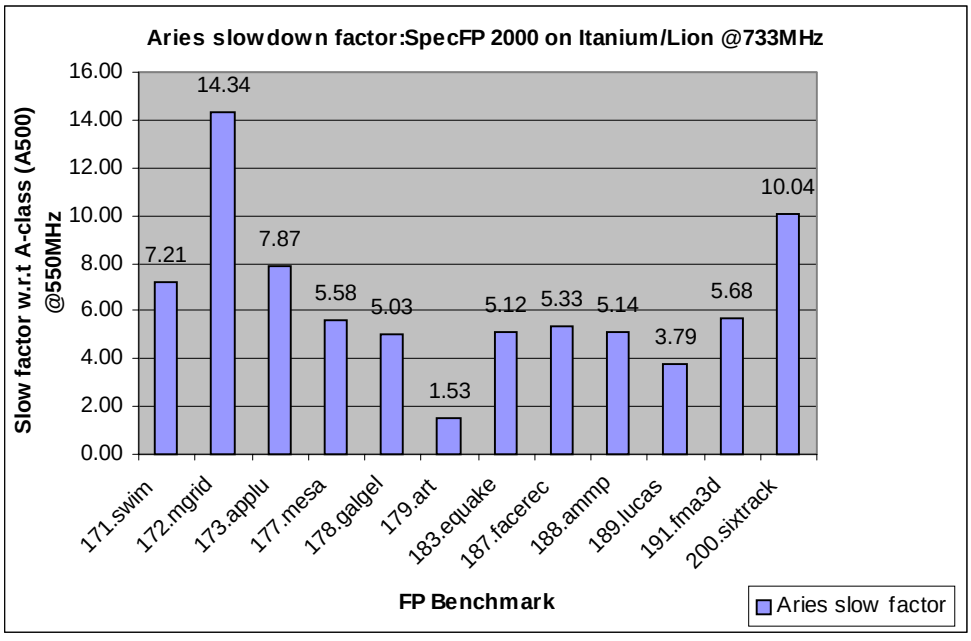

Figure 4. Aries Performance for Spec2000 Floating Point Benchmarks

Performance notes:

Many commercial applications run without visible slowdown under Aries. Aries' performance is very good for user-interactive applications. Aries Performance on compute-

intensive applications will vary from application to application: long running applications with good code locality will run well under Aries. Critical compute-intensive applications should be recompiled for Itanium for higher performance.

Limitations

1. Aries (both 32 and 64-bit emulator flavors) emulates only pure PA-RISC/HP-UX applications. This means, mixed-mode execution of PA-RISC/HP-UX and IPF/HP-UX shared libraries is not supported.
2. Aries does not support applications that have a near maximum usage of their virtual address space. This is because Aries consumes a small amount of virtual memory address space of application.

Conclusion

Aries provides a transparent and extremely reliable migration path from PA-RISC/HP-UX to IPF/HP-UX. Aries deploys a combination of fast interpretation and dynamic translation technology to emulate the PA-RISC instruction set architecture efficiently. It has been extensively tested for correctness of ISA emulation. Aries also faithfully simulates the PA-RISC/HP-UX system behavior to emulate all PA-RISC/HP-UX applications on IPF/HP-UX correctly. Aries performance with user intensive application has been comparable to that of PA-RISC/HP-UX systems. For compute-intensive applications Aries performance has been at acceptable levels.

References:

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