Running IA-32 Code on IA-64

Christophe de Dinechin
Hewlett-Packard Always-On Infrastructure Division
ddd@cup.hp.com

Abstract

IA-64, the processor architecture designed by Hewlett-Packard and Intel, has the capability to run a vast majority of the existing IA-32 (Pentium) code. This includes in particular most Windows or Linux binaries currently available. Today's versions of Windows and Linux for IA-64 offer this feature, in a way largely transparent to users. HP is also working to add similar support in HP-UX using a virtual machine. IA-64, however, is not merely a new IA-32; it is a new and different architecture. To run IA-32 code, IA-64 requires specific operating system or application support. In the long term, operating system support will be critical in getting reasonable performance for IA-32 code running on IA-64.

1. Introduction

When HP and Intel joined forces a few years ago to design the IA-64, they had different and somewhat conflicting needs. HP was looking for a cost-effective and high-performance successor to their PA-RISC architecture. Intel wanted a share of the profitable high-end segment of the market, largely dominated by proprietary RISC chips.

To meet the performance requirements and offer room for future growth in that area, the IA-64 architecture had to significantly break from the past. As is now well known, IA-64 is the first widely available explicitly parallel architecture, meaning that the compilers can give the processor multiple operations to perform in parallel. And explicit parallelism is only one of its many innovative aspects.

But breaking with the past has its drawbacks. Most notably, a new CPU architecture will not directly execute existing applications designed for older CPUs. Intel had a legacy of thousands of Pentium applications, while HP had to deal with a smaller, but still very significant amount of existing 32-bit and 64-bit PA-RISC code. Transitioning this valuable legacy to the new architecture was a key element of the IA-64 strategy, as illustrated on Figure 1.

HP took a software-based approach to supporting their legacy [1]. The HP-UX operating system for IA-64 ships

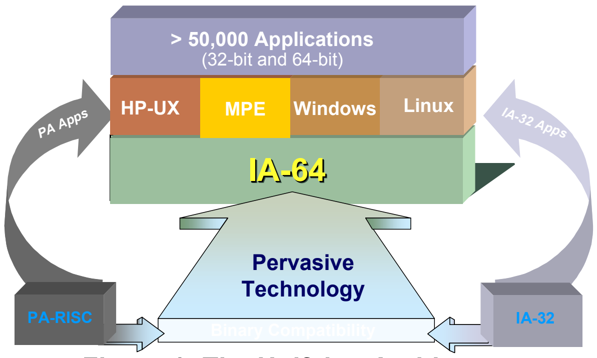


Figure 1: The Unifying Architecture

with software that dynamically translates PA-RISC code to IA-64 code on the fly, as illustrated in Figure 2. This translation process is totally transparent to users: they just run their existing 32-bit or 64-bit PA-RISC application on IA-64 as they would do on a PA-RISC machine, and they can freely mix and match IA-64 and PA-RISC applications.

Thus, HP-UX on IA-64 achieves real application binary compatibility with previous HP-UX versions on PA-RISC, although this binary compatibility is implemented in software. In practice, this approach takes advantage of the numerous similarities between PA-RISC and IA-64. It was also reasonable because HP alone controls the HP-UX operating system.

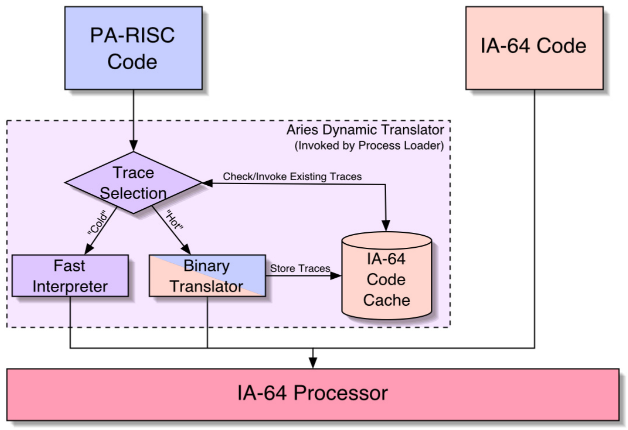


Figure 2: Transparent execution of PA-RISC binaries on HP-UX for IA-64

A similar approach would also have been technically feasible for executing IA-32 code on IA-64, and has been attempted for several other RISC chips [2]. On the other hand, the number of IA-32 operating systems and applications both suggested direct architectural support for IA-32 code. The IA-64 credibility as a successor for IA-32 would probably have been much lower without such direct architectural support.

In addition, Intel does not directly control any operating system running on their IA-32 architecture. Microsoft, the Open Source community, and many other partners deliver operating systems for Intel CPUs. Expecting each of these operating systems partners to build a binary translation technology was not realistic.

Instead, the IA-64 Architecture definition specifies the capability to execute most IA-32 code, as well as the possibility to mix IA-32 and IA-64 code, even in the same application [3] [4]. This support, however, is not enough to be able to run IA-32 applications on an IA-64 system. Significant operating-system support is still required to be able to run IA-32 binaries on an IA-64 operating system, as will be explained in Section 5.

Note that the Intel specification does not preclude a software-only implementation of this IA-32 support in future iterations of the architecture. As we shall see in Section 6, such support may make sense, since a software solution might outperform hardware implementations in a not so distant future. However, Itanium, the first implementation of the IA-64 architecture, supports the IA-32 instruction set largely in hardware.

In the rest of this article, we will describe how and why users may want to use IA-32 support on IA-64, and how HP plans to take advantage of that feature in HP-UX.

2. Why Run Existing IA-32 Code?

If you use an IA-64 system, why can't you just run native IA-64 applications? In particular, why not recompile all IA-32 applications to IA-64 to get the best possible performance?

The main reason is that recompiling an application, or building a different binary has a cost, and that cost may not be justifiable, at least initially. The cost of deploying an IA-64 version of an application includes: development time to make the application "64-bit clean", possibly rewriting the machine-dependent pieces (notably any assembly code), testing and certifying the IA-64 version, shipping a different physical medium, and having to support more than one platform.

Because of these costs, many software developers will develop only for the dominant platform. IA-32 is this dominant platform today and for the immediate future. Even in the most optimistic scenarios, IA-64 will not displace IA-32 in the next few years. This means that many applications will not be ported, or at least not immediately. They will remain available only for IA-32.

Windows Applications: This is particularly true for applications that would see no real benefit in running on IA-64. The kind of application that does not improve when ported to IA-64 includes in particular office applications such as Microsoft Office, since most users are not limited by CPU power for these applications. So we should not expect Microsoft to port their Office suite to IA-64 anytime soon: it simply does not make much business sense.

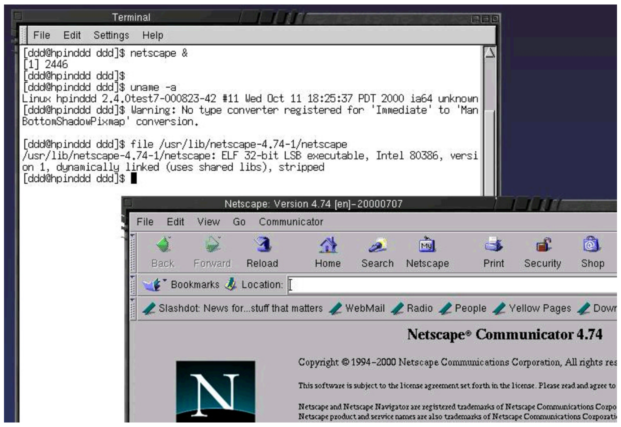
Similarly, thousands of indispensable Windows programs that are not CPU bound, like file transfer utilities, terminal emulators and a myriad of shareware or freeware would see no visible benefit if recompiled for IA-64. In many cases, the developers would not even consider buying the required IA-64 hardware to simply attempt the port. Therefore, this software is unlikely to become available in an IA-64 binary version in the near future.

Linux Applications: For Linux, the problem is slightly different. Contrary to Windows applications, Linux applications are most often given in source code form. And the experience gained by running Linux on other 64-bit platforms, such as Compaq Alpha, increase the chances that recompiling these applications for Linux on IA-64 will "just work." For this very reason, the initial releases of Linux for IA-64 already contain a vast majority of IA-64 code.

Yet, there are many applications that are available for Linux only in binary form. This includes very popular tools such as Netscape Communicator, office suites (ApplixWare), video games, or enterprise infrastructure applications such as HP's TopTools. Contrary to open source Linux code, which is generally mostly portable across processors, these applications are often only available for the Pentium processor, as any user of Linux on PowerPC or Alpha can testify. Usually, versions for other processors would be only "a simple recompilation" away. But even the minor effort of this recompilation does not occur.

OS Support: Since many IA-32 applications will not immediately be available in IA-64 binary form, being able to run IA-32 applications will be of significant value during the initial transition period. This is the reason why both Windows and Linux versions for IA-64 will offer support for their respective flavor of IA-32 applications. This makes it possible to run IA-32 Windows binaries on Windows for IA-64, and similarly to run IA-32 Linux binaries on Linux for IA-64.

Figure 3 illustrates the IA-32 version of Netscape Communicator for Linux running on the IA-64 version of Linux. Figure 4 shows various common PC applications running on a prototype of the IA-64 version of Windows.



```
[ddd@hpinddd ddd]$ netscape &
[1] 2446
[ddd@hpinddd ddd]$
[ddd@hpinddd ddd]$ uname -a
Linux hpinddd 2.4.0test7-000823-42 #11 Wed Oct 11 18:25:37 PDT 2000 ia64 unknown
[ddd@hpinddd ddd]$ Warning: No type converter registered for 'Immediate' to 'Man
BottomShadowPixmap' conversion.

[ddd@hpinddd ddd]$ file /usr/lib/netscape-4.74-1/netscape
/usr/lib/netscape-4.74-1/netscape: ELF 32-bit LSB executable, Intel 80386, versi
on 1, dynamically linked (uses shared libs), stripped
[ddd@hpinddd ddd]$ █
```

Netscape® Communicator 4.74

Copyright © 1994-2000 Netscape Communications Corporation, All rights res

This software is subject to the license agreement set forth in the license. Please read and agree to

Netscape and Netscape Navigator are registered trademarks of Netscape Communications Corpo
Netscape product and service names are also trademarks of Netscape Communications Corporati

Terminal
File Edit Settings Help
Netscape: Version 4.74 [en]-20000707
File Edit View Go Communicator
Back Forward Reload Home Search Netscape Print Security Shop
Bookmarks Location:
Slashdot: News for...stuff that matters WebMail Radio People Yellow Pages Down

Figure 3: IA-32 Netscape on IA-64 Linux

HP-UX users: HP-UX users are not used to executing IA-32 code, although various software emulation solutions have been developed with limited success. The use of IA-32 applications for IA-64 HP-UX users will most likely fall in one of two categories:

- Workstation users would typically use IA-32 applications for productivity (Office applications). This can alleviate the need for a second workstation or PC.
- Server users can use IA-32 support to transition existing server applications such as Exchange and consolidate then on IA-64 machines.

Actually, being able to run IA-32 code on IA-64 HP-UX machines is not a primary goal for HP. Instead, it is a positive side effect of a software partitioning solution that HP is developing to allow multiple operating systems to share CPUs and other resources.

Performance Limitations: IA-32 code executing on an IA-64 machine will not, in general, run as fast as on the latest and greatest IA-32 machine. This is discussed more in detail in Section 6. The bottom line is that you should not run performance critical IA-32 code on an IA-64 machine. If you need performance, use native IA-64 code.

3. How to Run IA-32 Code

Thanks to this integrated operating system support for IA-32 binaries in IA-64 versions of Linux and Windows, the process for running IA-32 applications is quite straightforward on both platforms. To the user, IA-32 and IA-64 applications behave largely identically. In other words, running an IA-32 application on both platforms is not really different than running an IA-64 application.

WOW64: For Windows, support of IA-32 code on IA-64 is quite similar in principle to support for 16-bit code in 32-bit versions of Windows. Just as there is a "Windows-on Windows" (WOW) layer in 32-bit Windows that executes 16-bit code in a sandbox, there is a 64-bit Windows-on-Windows (WOW64) that supports the execution of IA-32 code.

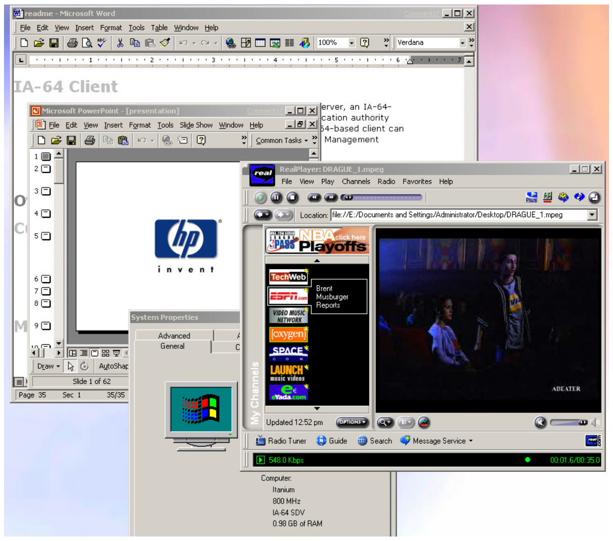


Figure 4: IA-32 Apps on IA-64 Windows

An IA-32 application runs on an IA-64 machine just like it would on any PC, as shown in Figure 4. Only a few minor differences are visible, mostly in the way applications are installed. Shared libraries containing IA-32 code cannot be used for IA-64 applications, and conversely. Practically, shared libraries containing IA-32 and IA-64 code are stored in different locations. This is reminiscent of the 16-bit to 32-bit transition, which saw the introduction of the `WINDOWS\SYSTEM32` for 32-bit code, alongside the traditional `WINDOWS\SYSTEM` directory containing 16-bit shared libraries.

On the IA-64 version of Windows[1], a similar solution is used. Interestingly, an empty `WINDOWS\SYSTEM` still appears. Two other directories in the `WINDOWS` directory are more interesting: `SYSTEM32 and SysWOW64`. At first sight, they appear to contain more or less the same list of files, including a large number of DLLs.

But sampling various DLLs indicates that they don't have the same size at all. For instance, `KERNEL32.DLL` is

1. This information is based on preliminary software, but should remain essentially valid on the released product.

approximately 800K in SysWOW64, versus approximately 1.6MB in SYSTEM32. As a general rule, almost all files are roughly twice as big in SYSTEM32 as they are in SysWOW64. The reason is that the files in SYSTEM32 are actually (rather counter-intuitively) IA-64 native 64-bit code, whereas the files in SysWOW64 are (again, a bit surprisingly) the classical 32-bit equivalents. IA-64 code is generally significantly larger than equivalent IA-32 code.

Applications also install at different places. 64-bit IA-64 applications install in `Program Files`, while IA-32 applications install in `Program Files (x86)`. In general, old 16-bit applications cannot install on IA-64.

Linux: When you launch a Linux application (for instance from the command line), Linux automatically detects if it is an IA-32 application, in which case it starts executing it in the IA-32 subsystem. Since the operation is done by the kernel, it is possible to mix IA-32 and IA-64 applications for instance in scripts, makefiles, or in client-server setups.

As with Windows, IA-32 and IA-64 shared libraries are stored in different places. IA-64 libraries reside in `/lib` or `/usr/lib`, while a directory under `/usr` contains separate `/lib` and `/usr/lib` for IA-32 libraries. This directory is called `/usr/glibc21-i386` on some distributions, but the name might change between Linux versions. IA-32 libraries are automatically picked up by the IA-64 version of the dynamic loader when running IA-32 applications.

HP-UX: Running IA-32 applications on HP-UX will involve one more step compared to other operating systems. HP-UX does not and probably will never support IA-32 applications natively. Instead, a software partitioning solution, or “virtual machine”, will be used to boot IA-64 operating systems such as Windows for IA-64 or Linux for IA-64. Such a virtual machine allows an operating system, called a “guest” to run as an application of another operating system. In effect, it partitions a CPU between operating systems in software, hence the other name of “software partitioning.” This is illustrated in Figure 5.

The IA-32 support of these guest operating systems will then be used to execute IA-32 applications. This solution is slightly less convenient than directly executing IA-32 code. On the other hand, it offers better performance and compatibility, since it reuses the compatibility layers designed by the Linux and Windows architects, rather than reinvent its own. More importantly, this solution is looking forward rather than backwards.

Mixing IA-32 and IA-64 libraries: Currently, it is not possible to have a single Windows or Linux application that uses both IA-32 and IA-64 shared libraries at the same time. The only exception are IA-64 shared libraries used by

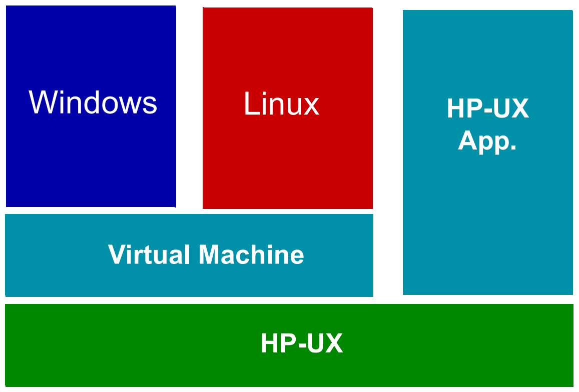


Figure 5: Virtual Machine

the system to allow execution of IA-32 code. We will discuss the reasons for this limitation in Section 5. In practice, this limitation imposes a duplication of many shared libraries, if these shared libraries have to be used both by IA-32 and IA-64 applications.

4. IA-64 Architecture Support for IA-32

How does IA-64 execute IA-32 code? As we said earlier, the IA-64 architecture specifies how IA-64 processors execute IA-32 instructions. To simplify slightly, IA-64 processors support application-level instructions in hardware, but offers little, if any, support of “privileged” IA-32 instructions, which would be used by IA-32 operating systems. A few IA-32 instructions also behave slightly differently than on a real IA-32 processor. This is described in more details in Chapter 6 of "*IA-64 Architecture Software Developer’s Manual Volume 1: Application Architecture*" [3], entitled “*IA-32 Application Execution Model in an IA-64 System Environment*.”

Instruction Set Transition: There are two ways to switch from one execution model to the other, as illustrated on Figure 6.

- The `br.ia` and `jmpe` instructions allow IA-64 and IA-32 code, respectively, to branch to code written for the other instruction architecture. These instructions allow applications to switch from one instruction set to the other.
- While IA-32 code is executing, any interruption will transfer control to the operating system and implicitly switch to IA-64 code execution. The return from interruption (`rfi`) instruction will switch back to IA-32 execution after the interruption processing is complete.

The instruction set transitions can be disabled using a specific bit in the processor status register. Thus, IA-64

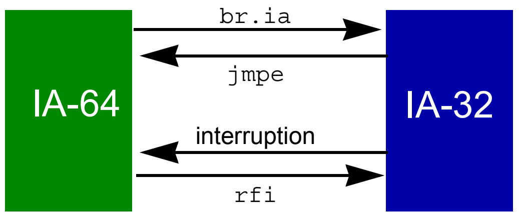


Figure 6: Instruction Set Transitions

operating systems need not all support IA-32 code execution. Even operating systems that allow execution of IA-32 code can select to disable instruction set transition within an application if they do not support mixed mode execution. In all cases, even when IA-32 application code is running, the operating system itself still executes using the IA-64 instruction set.

IA-32 Registers: IA-32 registers are stored in specific registers of the IA-64. For instance, IA-64 register GR8 holds the value that corresponds to EAX on the IA-32. This organization allows an IA-64 operating system to access the IA-32 state, for instance to save the state of an IA-32 process, much in the same way it would for any other part of the IA-64 state.

The IA-32 floating-point stack and the IA-32 MMX registers `MM0` through `MM7` are stored in IA-64 floating-point registers FR8 through FR15. Since these IA-64 registers are wider than their IA-32 counterparts (82-bit on IA-64 vs. 80-bit on IA-32), software may need to ensure that valid IA-32 values are stored in these registers before executing floating-point IA-32 code.

A few IA-64 registers are reserved to store values that are useful only to IA-32 code execution. These registers are not used by normal IA-64 code. For instance, application register AR24 contains the `EFLAG` IA-32 register. An operating system that does not directly support IA-32 code, such as HP-UX, may ignore these registers completely.

Last, many IA-64 registers are corrupted in an undefined manner when executing IA-32 code. These registers are used for IA-32 code execution. For instance, IA-64 registers GR1 through GR3 may be destroyed by the execution of IA-32 code.

Memory and I/O Accesses: In general, the IA-64 virtual memory model supersedes the traditional IA-32 segmented memory model when executing IA-32 code on an IA-64 machine. For instance, IA-32 uses a segmented 32-bit address space, while IA-64 uses a 64-bit flat address space. However, IA-32 code executing on IA-64 first converts all IA-32 addresses to a flat 32-bit virtual address, and then zero-extend it to a 64-bit virtual address, before using the IA-64 virtual memory to convert that virtual address to a physical address.

Thus, IA-32 code can only access a fraction of the whole 64-bit virtual address space. On the other hand, since IA-32 code only generates virtual addresses and the IA-64 virtual memory management remains fully in effect, even IA-32 code can access memory located anywhere in the 64-bit physical address space. This is shown in Figure 7.

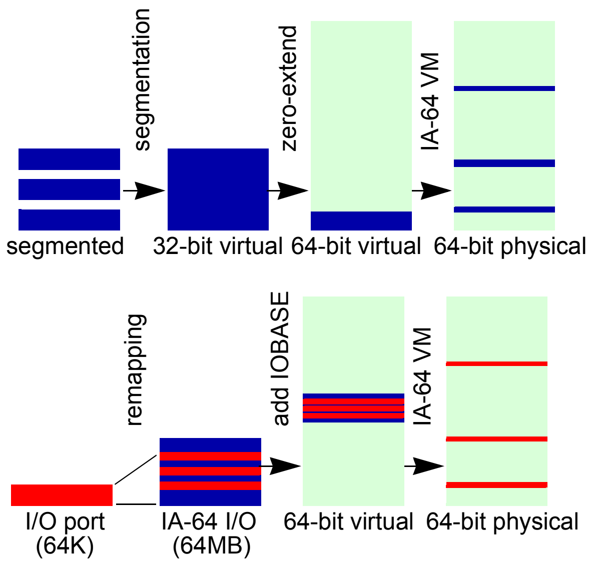


Figure 7: IA-32 Memory and I/O on IA-64

I/O port accesses are handled slightly differently. IA-64 has no notion of an I/O space, contrary to IA-32, which has special `IN` and `OUT` instructions. So IA-32 I/O operations are converted to normal memory operations. Thus, IA-64 implements "memory mapped I/O". The base of the IA-32 I/O space is specified with a dedicated register, KR0 (also called IOBASE.) The 64KB IA-32 I/O space then maps to a 64MB range in the IA-64 virtual memory space. The extension from 64K to 64MB allows different I/O devices to be mapped in different IA-64 virtual memory pages. This is also illustrated in Figure 7.

Since IA-64 virtual memory remains in effect for IA-32 code, all virtual memory attributes are taken from the IA-64 side, and any attempt to specify them from IA-32 code will cause a fault.

IA-32 Faults and Traps: An IA-64 operating system always executes IA-64 code. Therefore, it needs to intercept all interruptions that occur while executing IA-32 application code. Selected IA-32-related interruption conditions map to one of two reserved IA-64 interruption vectors (IA-32 Exception and IA-32 Intercept).

- IA-32 Exception is used for interruptions that exist on IA-32, but don't map correctly on their IA-64 counterpart. For instance, an IA-32 General Exception condition transfers control to the IA-32 Exception vector.

- IA-32 Intercept is used for interruption conditions that would not occur or might not occur on a real IA-32 hardware. For instance, executing an IA-32 privileged instruction invokes the IA-32 Intercept vector. Some conditions that would not trap on a real IA-32 machine can also be made to trap on IA-64,
- Other interruption conditions are treated identically whether they occur while running IA-32 or IA-64 code. This includes in particular most memory-related faults, which the IA-64 operating system is expected to handle identically whether the code causing the fault uses the IA-32 or IA-64 instruction set. In particular, IA-32 code may cause interruptions that do not even exist on IA-32 at all, such as TLB miss faults (IA-64 uses some software support for TLB misses, while IA-32 does it entirely in hardware.)

Other Differences: In addition to privileged IA-32 instructions, a few other instructions behave differently on IA-64 than they do on a "real" IA-32 implementation. In general, instructions that would affect the system state of an IA-32 machine may be configured to cause an IA-32 Intercept interruption on IA-64.

These instructions include `POPF`, `PUSHF`, `STI` and `CLI` (when they affect the interruption state), `CALL` and `RET` (when they affect privilege level), locking instructions, or instructions affecting the stack segment. Several branch instructions may also take an additional taken branch trap if the taken-branch-trap bit in the IA-64 processor status register is set.

With this support, it is possible to execute an IA-32 operating system on an IA-64 system, provided the various faulting conditions are intercepted and emulated by appropriate software.

5. Operating System Support

Since several registers are used differently by IA-64 and IA-32 code, operating systems need to be aware if IA-32 code is executing. A special bit in the processor status register indicates which instruction set architecture is currently in effect. An operating system that supports IA-32 code needs also to implement IA-32 specific interruption vectors, and to support a virtual memory layout for IA-32 applications, restricted to the first 4GB of the 64-bit virtual memory space.

But the most important aspect of IA-32 support is the availability of an environment suitable for execution of IA-32 code. IA-32 applications must be able to perform IA-32 system calls, to use IA-32 libraries, and more generally to use any IA-32 method to access the underlying system. This is illustrated in Figure 8.

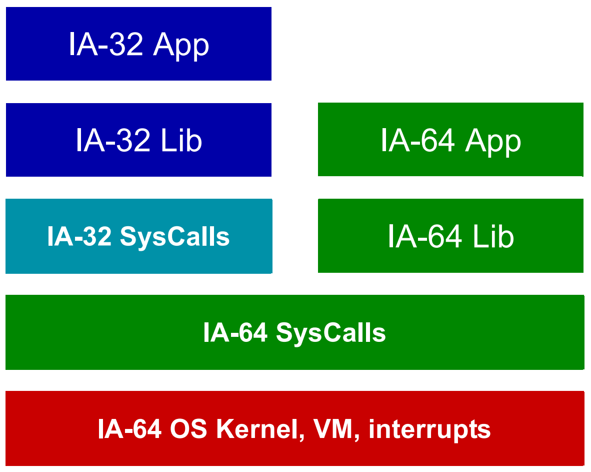


Figure 8: OS Support for IA-32 Code

Naturally, Windows for IA-64 supports the execution of Windows 32-bit applications, while Linux for IA-64 supports the execution of Linux IA-32 binaries. In theory, however, nothing would prevent for instance an IA-32 Windows subsystem such as Wine to be ported and run on IA-64 Linux.

Windows: On Windows, the WOW64 is implemented in the following components:

- `wow64.dll` provides stub functions to access for the NT kernel (`ntoskrnl.exe`) from IA-32 applications.
- `wow64win.dll` offers a similar service for the Win32 subsystem functions.
- `wow64cpu.dll` implements IA-32 emulation and IA-32 to IA-64 mode switch.

Most system DLLs found in `SysWOW64` are largely unmodified IA-32 versions. Only DLLs that share data with IA-64 DLLs are aware of the existence of WOW64. Another change is that the system call convention used on IA-32 is not used by WOW64. Instead, stub functions convert parameters from their IA-32 format to the IA-64 format in user space, and then perform a regular IA-64 system call. This method reduces the overhead of switching from user mode to kernel mode, and avoids the complexity of maintaining two conventions for entering the IA-64 kernel. And executing kernel code using the IA-64 instruction set and calling conventions maximizes performance.

Stub functions are implemented only for selected entry points. There is no general mechanism allowing an IA-32 library to call an IA-64 library or conversely. The reason is that the conversion of parameters and data structures (also called "arguments marshalling") can only be done if their types and layout are known in advance. Such conversion cannot be done for arbitrary, unknown functions. As a result, it is not possible to mix IA-32 and IA-64 code easily.

Linux: Like Windows, Linux maintains a separate path for IA-32 libraries. Some kernel support also exists, to translate IA-32 system calls to IA-64 equivalents. This support had some impact on the design of Linux for IA-64 as a whole:

- Linux uses identical system call, `errno` and signal numbers on IA-32 and IA-64.
- Linux for IA-64 by default reserves the first region ("region 0") of the virtual memory space for use by IA-32 applications.

The kernel support for IA-32 Linux binaries on IA-64 Linux can be found largely in the `arch/ia64/ia32` source directory of Linux. Most of the work there is to convert parameters from their IA-32 format to the corresponding IA-64 format. Contrary to Windows, this is done mostly in kernel space.

In many cases, only some very simple conversions need to happen (mostly to convert from IA-32 calling conventions to IA-64 calling conventions), and the rest of the system call path is then identical with an IA-64 system call. In a few cases, more significant conversion is required. For instance, the IA-32 version of `ioctl()` recognizes and converts many types of `ioctl()` arguments. As we said earlier, there is no general method for converting arguments from IA-32 to IA-64 format, so "private" `ioctl()` values implemented by some rare custom drivers would not get converted correctly.

HP-UX: In the case of HP-UX, no specific OS support for IA-32 code exists. HP-UX "32-bit support" is much geared towards supporting binary or source compatibility with PA-RISC 32-bit applications. For instance, region 0 in the virtual memory space is reserved for 32-bit applications or for sharing data between 32-bit and 64-bit applications.

The work that is being done in HP-UX with respect to IA-32 code execution is limited to what is needed to support a software partitioning solution and to run multiple IA-64 operating systems simultaneously on a single CPU.

6. Performance

Even if it is implemented in hardware, IA-32 support on early IA-64 implementations is far from being on a par with the most recent implementations of IA-32 such as the Intel Pentium IV. What's more, since IA-64 is so different from a traditional IA-32 CPU, the performance results will vary dramatically depending on the application.

In general, the user experience running IA-32 code can be compared to that of a 200MHz to 300MHz Pentium system. But the performance of a given application is very difficult to predict.

On one hand, a ray tracing application that we tested ran about 15 times slower than on a 650MHz laptop. On the other hand, Quake produces very acceptable frame rates. Microsoft Word and other Office applications are completely usable, but show noticeable delays displaying some graphic types. RealPlayer runs MPEG video smoothly, but some other players won't run at all. Even time-honored benchmarks like SPEC do not show much more consistency across individual tests. Some traditional Windows benchmarks fail to install, and the error message indicates that this is because of 16-bit code.

Providing point performance data is therefore almost meaningless, even less so considering that the software and hardware we used was still prototype at the time we conducted our experiments. A more interesting analysis, based on experience using various applications and recompiling various performance kernels, is to try to understand what really impacts performance of IA-32 code running on an IA-64 machine. Since this is based on early software and hardware, the picture may still vary a lot in the months or years to come.

Figure 9 illustrate how various kinds of applications may behave, based on their respective usage of the processor. As this figure shows, there is no general rule, and applications may exhibit many different behaviors.

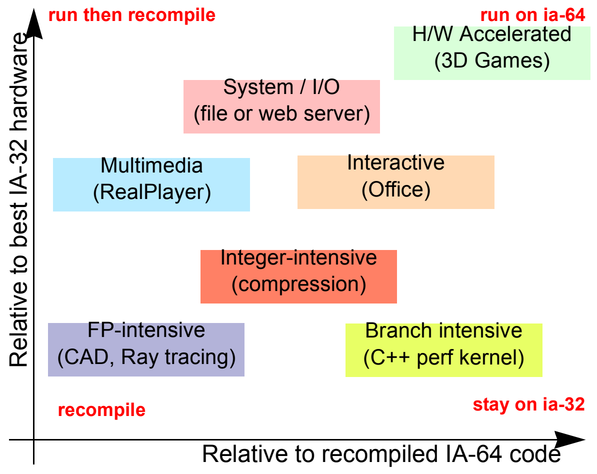

run then recompile
run on ia-64
H/W Accelerated (3D Games)
System / I/O (file or web server)
Multimedia (RealPlayer)
Interactive (Office)
Integer-intensive (compression)
FP-intensive (CAD, Ray tracing)
Branch intensive (C++ perf kernel)
recompile
stay on ia-32
Relative to best IA-32 hardware
Relative to recompiled IA-64 code

Figure 9: IA-32 Performance Scenarios

In general, the factors that seem to influence performance the most include:

- Does the application spend its time in user or kernel space? Since kernel code executes at full IA-64 speed, kernel-intensive applications will generally behave better.
- How does the application use memory? IA-64 caches and memory subsystem are designed to take advantage of compiler hints (prefetching, speculation). Cache

locality in particular will play a larger role on IA-64 than for a native IA-32 CPU.

- Does the application use floating-point? In our experience, IA-32 floating-point applications perform poorly, whether compared to a native IA-32 machine or to recompiled IA-64 code. If you have a floating-point intensive application, chances are you want to recompile it.
- Does the application use indirect calls a lot? In particular, some C++ applications, where performance of virtual function calls matters, may see a significant impact, since dynamic branch prediction is very good in recent IA-32 implementations. This is one case where recompiling for IA-64 did not yield much improvement. Compiler-directed branch prediction is still in infancy, in particular on Linux.

In conclusion, the most reasonable guidelines to select which IA-32 applications will run reasonably on IA-64 systems is: try it!

7. Long-Term Vision

IA-32 code is not going away. For IA-64 to be successful, it has to keep up with IA-32 performance for as long as IA-32 remains a major player. Supporting IA-32 code is only the first and necessary step in this process. But it is unreasonable to dedicate ever-increasing silicon space to this IA-32 support. IA-64 will never execute 32-bit code as well as a dedicated processor, just as the Pentium Pro never executed 16-bit code as well as original Pentium chips.

Therefore, the success of IA-64 as a successor to IA-32 will depend on three factors:

- The native performance of IA-64 needs to increase regularly, both because of faster chips and because of improvements in compiler technology. If and when IA-64 applications significantly outperform IA-32 code, performance-critical software will naturally migrate to IA-64, and the importance of IA-32 performance will decrease.
- IA-64 must enable CPU technology developments that would be impossible or too costly with IA-32 or RISC chips. If and when IA-32 performance gains become marginal despite massive investments, IA-64 will become a natural successor.
- Dynamic translation technology must be developed, that generates IA-64 code from IA-32 code on the fly. A similar technology allowed Apple to replace the Motorola MC68000 with the PowerPC, and in less than two years to best any silicon implementation for MC68000 code performance. Hopefully, IA-64 will soon become the fastest way to execute IA-32 code.

HP is taking the necessary steps and investments one at a time. Today, we are positioned with IA-64 in the high-end markets. But we are already developing technologies that will matter to you when you will install an IA-64 on your desktop and want to run all your old applications.

8. Conclusion

For HP-UX users, virtual machine technology offers the prospect to run multiple operating systems on their machine at the same time, and for the first time, to gain access to numerous Windows and Linux applications while preserving their investment in PA-RISC and HP-UX code.

With this approach, HP both increases the value proposition of IA-64 and ensures smooth transition for all its customers. Virtual machine technology is a perfect illustration of the new “Three OS strategy” that HP embraced in the last years.

9. References

[1] *PA-RISC to IA-64: Transparent Execution, No Recompilation”.* Cindy Zheng, Carol Thompson, IEEE Computer Society Cover Feature, 3/2000
http://computer.org/computer/co2000/r3047abs.htm

[2] *Digital FX!32: Combining Emulation and Binary Translation.* Raymond J. Hookway and Mark A. Herdeg
http://www.digital.com/DTJP01/DTJP01HM.HTM

[3] *IA-64 Architecture Software Developer's Manual Volume 1: Application Architecture.* Version 1, July 2000
http://developer.intel.com/design/ia-64/downloads/24531702s.htm

[4] *IA-64 Architecture Software Developer's Manual Volume 2: IA-64 System Architecture.* Version 1, July 2000
http://developer.intel.com/design/ia-64/downloads/24531802s.htm

[5] *Intel Architecture (IA-32) Software Developer's Manual, Volume 3: System Programming.* Intel, 1999
http://developer.intel.com/design/pentium4/manuals/245472.htm

Christophe de Dinechin is working in the Always On Infrastructure Division as software architect for HP-UX virtual machine technology. In previous positions at HP, he worked on rehosting and retargetting HP's C and C++ compilers to IA-64, and designed a real-time test executive software for HP Test and Measurement (now Agilent Technologies.) He can be reached at `ddd@cup.hp.com`.