# Linux on PA–RISC
# One Martini Too Many

Matthew Wilcox
matthew@wil.cx

## Abstract

Hewlett-Packard's PA–RISC processor is one of the few remaining mainstream processors without Linux support. This talk outlines the progress of the port to date, including a brief rundown of the available hardware, as well as a description of the basic architecture.

We will detail kernel changes required for the PA–RISC CPU, particularly those required for SMP support and PCI cards. Linux/PA–RISC includes an HP–UX emulation layer, which we will also discuss. Most of the work involved in porting to a new architecture involves supporting all the devices available. PA–RISC is no exception. Even though drivers were available for many of the chips used, work was still required to get them to function properly.

The pre-existing HP–UX support in GCC and binutils was suited for the 32 bit SOM format. The changes to the toolchain, which were required to support ELF, will be covered. The GNU C Library also had to be ported to the PA–RISC processor.

## 1 Introduction

I first got involved in the PA–RISC project in May 1999 at Linux Expo in Raleigh, North Carolina. I found an HP Apollo 715/33 being used as a monitor stand at work and wondered if I could put it to use. I found the Mach port, but the 715/33 was the only workstation in its class that was unsupported. I heard about the port being organised by The Puffin Group shortly before the Expo and agreed to help. I wrote a SOM executable loader for Linux and adapted the HIL keyboard driver originally written for the m68k port to work with PA–RISC. I accepted a job offer from The Puffin Group in August and started working on the PA–RISC port almost full-time in December.

## 2 PA–RISC Hardware

### 2.1 Processors

The PA–RISC processor is an HP-designed CPU. Even though both HP–UX and MPE/iX run in big-endian mode, some CPU models are switchable. It is slightly unusual in that the ABI defines the stack to grow upwards instead of downwards (though this is not intrinsic to the processor design). As with most RISC processors, it has a direct implementation of the instruction set (no microcode), 32 bit instructions, a small number of addressing modes and a load-store architecture. Almost all models have an integrated high-performance floating point unit.

The PA–RISC line has evolved over the past 15 years and produced the following models:

- PCX, the first CPU, implements the PA 1.0 architecture. It has a number of restrictions that make it a very hostile architecture on which to run Linux. Further, it was only available in some early servers, which have almost all been scrapped by now, so we have no plans to support it.
- PCX–S, PCX–T and PCX–T', which are older implementations of the PA 1.1 architecture. They are 32 bit processors and run at 48 to 120MHz.
- PCX–L and PCX–L2, which are newer implementations of the PA 1.1 architecture. They are also 32 bit processors and were an effort to

produce a low–priced design by integrating the GSC bus interface into the processor — this reduced chip count on the motherboard. Their clock speeds range from 60 to 180MHz.

- PCX–U, PCX–U+, PCX–W and PCX–W+, which implement the PA 2.0 architecture. They are 64 bit processors (which still support the full 32 bit instruction set) and run at 120 to 550MHz. All new HP machines use PCX–W or PCX–W+ processors.

## 2.2 Devices

Fortunately, HP have mainly used commodity components in the machines that interest us most. The Ethernet chip used is either an Intel 82596 for 10Mbit ethernet, a DEC Tulip for 100Mbit Ethernet or an Alteon for Gigabit Ethernet. For SCSI, HP have always used NCR–series chips (even though the chip design has been owned at various times by Emulex, NCR, Symbios and now LSI Logic). The serial ports are mainly 16450–compatible, with one exception mentioned below. HP continued using the HIL keyboard & mouse that was used with the series 300/400 m68k based workstations. Starting with the 712, the PS/2 keyboard & mouse were used instead, in order to reduce costs. This continued to be used in first generation B/C/J class workstations. The latest generation of B/C/J class workstations (eg C3000) have two USB ports for keyboard & mouse.

All machines support at least one proprietary HP bus, which has evolved from CIO through HP–PB (AKA NIO), SGC and GSC into HSC. All recent machines also support PCI and some older machines support an EISA bus.

There are several bus adaptors for which drivers had to be written. All the machines have a proprietary bus from the CPU to a bus adapter which mediates with the devices. On some machines, there is then a second layer of bus adapters.

For example, in the C3000, the CPU sits on the Runway bus. The System Bus Adapter (called Astro) interfaces the Runway bus to 8 Ropes busses. The PCI bus is attached to the Ropes bus through the Lower Bus Adapter (called Elroy). The extra level of bus converters allows the Runway bus to remain shorter and thus run at higher clock frequencies. So, despite the additional IO latency, the system bus can achieve higher throughput. The Astro/Elroy combination supports up to 8 64 bit, 66MHz PCI busses in a single system.

## 2.3 PA–RISC workstations

The original workstation series were the 705, 710, 720, 730 and 750. These machines are not numerous, and development tends to be neglected. They use the PCX–S CPU.

They were replaced with the 715/Scorpio, 725, 735 and 755. All these machines use the HP SGC bus, and have PCX–T series processors. They are moderately popular and are worth supporting.

The 712 and 715/Mirage use the PCX–L processor and have a GSC bus. There are more 712–series workstations than any other model of PA–RISC workstation.

After this, new workstations were given B/C/J letters. The C100, C110, J200 and J210 are PCX–T' based and have a Runway bus that has Runway to GSC bus converters on it. They also have GSC to EISA bus adapters.

The B132L, B160L, B180L, C160L and C180L are PCX–L2 based machines, which have the processor directly attached to the GSC bus instead of bridged from Runway. They contain a PCI bus that is bridged from the GSC bus via the Dino bus converter.

The C160, C180, J280 and J282 were the first 64 bit PA–RISC workstations. They have a PCX–U CPU on a Runway bus, bridged to the GSC bus. Their GSC bus has both an EISA and a PCI bridge on it.

They were superceded by the C200+, C240+ and J2240 workstations based on the PCX–U+ chip. These machines have both 32 and 64 bit PCI slots, but no EISA slots. They use the Dino GSC to PCI adapter for their 32 bit slots and the Cujo GSC to PCI adapter for their 64 bit slots. The C360 workstation replaces the PCX–U+ processor with a PCX–W, but is otherwise unchanged.

The B1000, B2000, C3000, J5000 and J7000 are workstations based on the PCX–W processor and have a Runway bus bridged to a PCI bus for expansion slots using Astro and Elroy (as discussed above). The C3600, J5600 and J6000 are similar, but use a PCX–W+ processor instead.

### 2.4 PA–RISC servers

The earliest PA–RISC machines were servers, and they have HP proprietary devices attached to HP proprietary bus architectures. It is unlikely that documentation on these busses and devices will ever become available, since so few people are interested in spending any effort finding and releasing it. Machines in this category are the E, F, G, H and I class machines (sometimes known as Nova) as well as some earlier, unlettered servers.

The D, K and R class machines may well become (at least) partially supported at some point in the near future. Architecturally, they are reasonably similar to the early C class machines. The T class is unlikely to be supported, as there seems to be very few accessible machines. The V class is HP's top-end machine, and has a very different architecture. We're looking at it longingly, but realistically, we are at least a year away from being able to run on that machine.

The A180 was the machine that HP first sent out to developers. It has a PCX–L2 processor on a GSC bus which is attached to a PCI bus by the Dino GSC to PCI converter. It is remarkably similar to the early B class systems.

The L class, N class, A400, A500 and A550 servers are based on the PCX–W or PCX–W+ processor, like the most recent B/C/J class workstations, but they have to be run in 64 bit mode as their PDC (the equivalent of the BIOS) does not support 32 bit operation. They also have a Runway bus and Astro/Elroy PCI bridges.

## 3 PA–RISC Quirks

### 3.1 Virtual Memory

The PA–RISC processor has a virtually indexed, physically tagged cache. Initially, we thought we needed to implement page colouring in order to have correct behaviour. Philipp Rumpf wrote a quick page colouring implementation. It turned out we were incorrect. In fact, the cache flushing macros used in the VM system are sufficient to give correct behaviour with the PA–RISC processor. The page colouring was removed as it was no longer necessary. The search for a good page colouring implementation for Linux continues, but that is the subject of an entirely different talk.

### 3.2 SMP

One of the major shortcomings of the PA–RISC instruction set is a lack of generalised atomic operations. In contrast to the Intel x86, which provides many complicated atomic operations, or Alpha, MIPS & PowerPC, which provide Load Locked / Store Conditional instructions to build arbitrarily complex atomic operations, the PA–RISC has only Load-And-Zero.

This presents quite a challenge as, for example, Linux uses a macro called `xchg()` which exchanges two values atomically. In order to have this work safely on SMP (and on a uniprocessor machine with interrupts enabled), we actually need to spinlock inside the implementation of `xchg()`.

We've considered multiple possible implementations of this, including one particularly perverse one I devised (which would be UP only) that checked in the interrupt handler whether it had interrupted a critical section and jumped back to the start of it if it had. In the end, we decided that saving one instruction here was not worth the additional complexity in the interrupt handler.

There are several other atomic operations we can't do efficiently, such as `atomic_inc_and_test()` and `test_and_set_bit()`. Again, we have to spinlock within these implementations. This is a sizeable obstacle, because the granularity of locks within the kernel is being reduced to the point where driver writers are encouraged to use these atomic operations instead of grabbing one lock. But on PA–RISC, using this strategy results in the CPU spending a lot of time acquiring and releasing locks, and disabling then reenabling interrupts with no benefit.

There is hope. Some machines appear to implement a variety of atomic operations in the memory bus adapter. We're waiting for HP to provide documentation on these.

## 3.3 Cache Coherency

On the Intel x86 architecture, the DMA controller interacts with the CPU caches (called cache snooping) to ensure that the CPU and devices have a coherent view of the memory contents. They can do this because the cache is physically indexed. PA–RISC caches are virtually indexed in order to enable CPU cache coherency checks to start before the physical address is looked up. The drawback to this design is that the coherency checker can't "see" physical accesses by a DMA device to memory. The problem for drivers is the data the device was supposed to read from memory may still be in the processor cache, rather than in the memory location that the DMA controller attempts to access.

Fortunately, Dave Miller of Linux/SPARC fame proposed the Dynamic DMA Mapping interface which solves this problem for PCI device drivers. Many PCI device drivers that do DMA have now (as of Linux 2.4.0-test1) been converted to use this interface. This includes all the drivers we care about, such as the Tulip ethernet driver and the Symbios SCSI driver. A full description of this interface is given in `linux/Documentation/DMA-mapping.txt`.

On machines that have an I/O MMU (PCX–T' or later), the Dynamic DMA Mapping interface is used to tell the I/O MMU that an address range is about to be used for DMA. The I/O MMU will provide virtual address tags for that address range as if the DMA device had used a virtual address. Thus, it participates in the CPU cache coherency protocol under software control.

The PCX–L and PCX–L2 based machines do not have an I/O MMU, but they do support uncacheable pages. This is a reasonable alternative for implementing the Dynamic DMA Mapping interface. When the memory is marked as uncacheable, the processor does not cache reads or buffer writes, meaning the main memory is always up to date. This has a certain performance penalty, so it is not as good as having an I/O MMU, but it does guarantee correctness.

The PCX–S and PCX–T based machines also do not have an I/O MMU, but they do not support uncacheable memory, either. These machines cannot implement the Dynamic DMA Mapping interface. Fortunately, none of these machines support PCI, so we don't need to modify the PCI device drivers. For these machines, the device driver has to explicitly flush the caches after writing to memory before initiating DMA. It also has to purge the caches after performing DMA, before reading from the memory. Some device drivers are already written in this way to support some Motorola 68000 designs that do not support cache snooping either.

## 3.4 I/O Interrupts and IRQ Regions

Unlike the x86 architecture, there is no physical IRQ line running to the processor. An I/O interrupt is generated by a write transaction to the CPU mastered by either a bus adapter (eg Dino) or I/O Device (eg Bluefish, a GSC 53c720 based SCSI card). Two problems make IRQ line emulation through a simple table infeasible.

The first is scalability. The EIRR (External Interrupt Request Register) in each processor stores the value of the data written in the transaction. Each bit in the EIRR represents the value of a pending interrupt. The OS assigns particular bits to individual devices/drivers. A 32 CPU machine would have to allocate many more IRQ "lines" than a single CPU machine and this would take up much more memory.

The second problem is hierarchy. The PCI bus adapter (ie Dino or Elroy) is just an agent for the actual PCI device which uses an IRQ line. Thus, another layer of software must provide service to the bus adapter in order to translate IRQ "line" status into a call to an Interrupt Service Routine (ISR).

Philipp Rumpf and Grant Grundler hammered out a design called IRQ Regions to handle the two main problems above. This design provides services to associate bits (EIRR or IRQ "line" status bits) with handlers (PA External Interrupt handler or PCI device ISR). It uses a virtual interrupt number for the benefit of the rest of the system which expects an interrupt to be represented by an integer. Layering the interrupt handlers gives us the hierarchy we require; each layer in the calling stack uses a unique IRQ Region. For example, the iosapic (Elroy) driver registers an ISR with the CPU IRQ Region while PCI devices below that Elroy must register with the IRQ Region for that instance of the iosapic.

PCI device IRQ bits are associated with the appropriate IRQ Region during the "Bus Fixups" phase of the PCI bus walk. This is after a PCI device is discovered but before the `struct pci_device` is made visible to PCI drivers. GSC device drivers use a different interface since those drivers program and master their own interrupt transactions.

## 4 Userspace

With progress on the kernel well underway, some people turned their attention to userspace. The kernel can already run some HP–UX executables, including HP–UX sh and many other basic utilities. To make a Linux userspace, getting GNU libc working was the top priority. Before we could do this, it was necessary to implement ELF support for PA–RISC.

HP–UX uses their own SOM format for 32 bit binaries, but uses ELF for 64 bit programs. We were building the kernel as a SOM image and executing HP–UX SOM binaries, but glibc requires a more sophisticated object format. The kernel is also more dependent on ELF features these days, particularly for exception handling, loadable modules and freeing initialisation memory.

HP had already employed Cygnus to supply a 64 bit ELF toolchain for HP–UX. We were able to build on that work to produce a (rather hackish, admittedly) 32 bit ELF toolchain, which was able first to compile the kernel and then later compile glibc.

Around this time, we had to decide on a distribution to port, as none of us were interested in starting a new distribution (or doing the work involved in keeping up to date with new releases of many pieces of software). We considered several distributions that were already multiplatform (so they had the necessary infrastructure to support another port) and those with which we had personal experience. After a certain amount of passionate discussion, we settled on Debian as our preferred distribution.

There are several good reasons for choosing Debian. It is the most widely-ported distribution, and has already solved endian and 64 bit issues. The package maintainers system has advantages for us, as individual maintainers can take care of feeding any PA–RISC–specific changes to the upstream version of their package. As a non-commercial organisation, Debian doesn't have business goals that could potentially conflict with those of HP.

## 5 Status

We're keeping up with Linus' changes to the 2.4-test kernels; most of the 700-series workstations are now almost fully supported by the kernel.

Work is still ongoing on the frame buffer device. Similarly, the floppy drive is not yet supported (on the few workstations that actually have one).

We have problems with the National Semiconductor chip that implements the serial, USB and IDE ports in the latest generation of B/C/J class workstations. This may be difficult to resolve within the current Linux PCI infrastructure.

A 64 bit toolchain is currently being prepared so that we can build a 64 bit kernel. More toolchain work is needed to support shared libraries and kernel modules. The SMP support in Linux/PA–RISC is still mostly theoretical. Much of the kernel is still very inefficient in its use of caches; this will be addressed in the coming months after we have a working system, we can spend more time profiling and optimising.

In userspace, we're currently focussed on building an installable Debian base system. We have a number of tarballs of useful basic programs available for download, but we're still some months from being able to `apt-get install` the system.

## 6 Acknowledgements

This port is very much a collaborative effort. Many people have contributed a lot of their time working on the project, including Paul Bame, Grant Grundler, John Marvin & Matt Taggart of HP; Alex deVries, Richard Hirst, Alan Modra & Martin Peterson of Linuxcare, Thomas Bogendörfer, Helge Deller & Philipp Rumpf. Other significant contributors include Alan Cox, Sam Creasey, Ulrich Drepper and Jeff Law.

## 7 References

This paper is available at
`ftp://puffin.external.hp.com/`
`pub/parisc/docs/ols2.tex`

More information is available from
`http://www.thepuffingroup.com/parisc/`
`http://www.debian.org/ports/parisc/`
`http://parisc.workstations.org/`
`http://docs.hp.com/hpux/systems/`
`http://devresource.hp.com/`