

Linux/ia64: Preparing for the Next Millennium

David Mosberger

Hewlett-Packard Labs

davidm@hpl.hp.com

http://www.hpl.hp.com/personal/David_Mosberger/

ABSTRACT

The IA-64 architecture has been co-developed by HP and Intel is intended to become the commodity computing platform for the next millennium. Interest in IA-64 has been widespread and has resulted in endorsements from all major players in the computing industry, including (in alphabetical order) Compaq, HP, Intel, SCO, SGI and Sun. For competitive reasons, the full architecture will remain a closely guarded secret until IA-64 based systems are generally available. This creates a bit of a dilemma for the open source community because it would mean that Linux development could not start until systems are widely available. To alleviate this situation, researchers at HP Labs in Palo Alto investigated what could be done to help accelerate Linux development such that it could be available from day one of the IA-64 computing era. This paper reports on their endeavors, accomplishments, and future plans.

1 Introduction

The HP/Intel co-designed IA-64 architecture will see first silicon with Intel's Merced chip. This chip is slated to appear in systems in mid-2000 and will quickly be followed by the second-generation, faster McKinley chip [3]. What is interesting about IA-64 is that it is not just a new architecture but that it introduces a

whole new computing paradigm called EPIC. This acronym stands for "Explicitly Parallel Instruction Computing" and implies that it breaks the barrier of traditional CISC or RISC [5] sequential programs by explicitly exposing instruction-level parallelism. The idea behind doing this is to make it possible to build extremely high-performing computers by running relatively simple circuitry at very high speeds and executing as many instructions as possible in parallel. EPIC, and with it IA-64, builds on the lessons learned from both RISC and very-long instruction word (VLIW) computers. For example, like in VLIW computers, IA-64 groups instructions into bundles that can be executed in parallel. Like RISC computers, each instruction is relatively simple and has a fixed length.

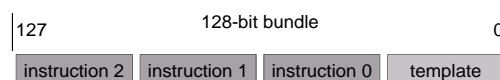


Figure 1: IA-64 Instruction Format

To give a more concrete feeling of what IA-64 is like, Figure 1 shows the instruction format. As illustrated, instructions are grouped into 128 bit wide bundles that each contain three instructions and a template field. The template field indicates which instructions in this bundle and in following bundles the CPU can execute in parallel. Each instruction is about 40 bits wide and contains a 6-bit predicate register field and three 7-bit general purpose register fields. This means

that an instruction can reference as many as three operands in a register file containing as many 128 different registers. The predicate field determines which of 64 predicate registers controls whether or not the instruction should be executed. This instruction predication support allows to conditionally execute code without having to resort to expensive branches. Another advanced feature of IA-64 that has been disclosed already is speculative loading. Load speculation allows to safely execute a load instruction well before it is known whether or not the loaded value is really needed. On traditional architectures this cannot be done because the load instruction may, for example, cause a segfault due to dereferencing a NULL pointer. Since memory accesses are comparatively slow on modern computer systems [6], speculation allows to hoist load instructions to a place far in advance of the place where the loaded value is being used. When used properly, this allows the CPU to waste less time waiting for the memory system and hence execute much faster.

While this brief introduction certainly cannot do justice to the IA-64 architecture, it does serve to highlight some of the more important features. For a more detailed description of the features mentioned above, the reader may want to read some of the cited materials [4, 2, 1].

2 The Goal

The goal of the Linux/ia64 project at HP Labs is to ensure that Linux will be available and run on IA-64 based systems from day one on. At a minimum, we hope that by the time IA-64 systems become available, Linux will be self-hosting such that other developers can join and start contributing enhancements and additions without having to depend on any non-free software. If the project goes well,

the state of affairs will hopefully have progressed substantially beyond this minimum goal. For example, it may be possible that some time will have been spent on optimizing performance critical parts of the system such as the compiler, math libraries, and key parts of the kernel. In any case, it should be made clear that the goal is emphatically *not* to create an “HP version” of Linux but rather to lay the foundation for forming a Linux/ia64 community much as it exists today for x86. For example, while the developers at HP will of course use HP development machines, the goal is to create a Linux system that will run on any IA-64 platform that adheres to the IA-64 platform architecture specification. In the same fashion, we hope to engage other members of the Linux community as early and as broadly as possible to ensure the project is true to the spirit of the Linux community.

3 What’s Involved?

Given the features of IA-64 and the goals described in the previous sections, what does it take to bring the Linux/ia64 project to fruition? Obviously the Linux kernel itself needs to be ported but, while essential, it is just one of the many pieces that need to be put in place. Indeed, before any work on Linux/ia64 can begin, there needs to be an IA-64 version of the GNU compiler. Both gcc or egcs could serve in this role, but non-GNU based compilers would not be acceptable because both the Linux kernel and GNU libc rely heavily on GCC-specific extensions to the C language. For convenience, it is desirable to have not just a GNU compiler but a complete GNU-based toolchain, including the GNU assembler, BFD object file support, and the GNU linker. Since actual IA-64 hardware will not be available for a portion of the project, it is also necessary to have a simulator that is sufficiently fast and accurate

enough to support both kernel and user-level development.

With these tools in place, kernel development can commence. However, even once the kernel is up and running, the project is far from over. At the user-level, it is first necessary to create an IA-64 version of the GNU C library and then the hundreds and thousands of common Linux utilities need to be built and tested. Since Linux is already 64-bit clean (for the most part), this is expected to largely entail simple recompilation. However, there are some important exceptions. For example, the GNU source level debugger (gdb) is platform-specific and will require some time and effort to port. Similarly, the X Window System is likely to require some IA-64 specific changes as well or will at least benefit from IA-64 specific optimizations. Finally, there is of course a big difference between a system that works and one that works *well* and fast. Hence, we expect that a fair bit of time will have to be spent optimizing performance-critical aspects of the system. Since EPIC relies heavily on compilers for achieving good performance, realizing the necessary optimizations in egcs is expected to take the largest amount of time. The good news here is that once a working version of egcs exists, compiler development is entirely independent of all the other development projects. As improvements are made to the compiler, it is simply necessary to recompile the relevant programs—no changes to already existing code will (should) be needed.

4 Status

The Linux/ia64 project started at HP Labs in February of 1998. Since a functional GNU C compiler is on the critical path to all other development work, we decided to work on it first. Actually, this is not the entire truth because it was felt that the compiler is too

big a piece of work to start the project with. Thus, we started out by setting ourselves the more modest goal of creating an IA-64 version of the GNU binutils, which includes ELF object file format support, the GNU assembler, linker, and disassembler. Since EPIC-style computing has features such as instruction bundles that are not found in other architectures, this task was not entirely trivial either, but it was certainly less complex than the compiler and allowed us to have some tangible results quickly.

By mid-May binutils was in good enough shape that work on an IA-64 backend for the GNU compiler could begin. It should be noted here that writing a well-optimizing compiler for IA-64 is a complex undertaking requiring non-trivial changes to the compiler infrastructure. In other words, turning the GNU compiler into something that is well-suited for generating code for EPIC-style architecture requires much more than a straight-forward IA-64 backend. For logistical reasons we were in no position to make such substantial changes to the compiler infrastructure (which is largely maintained and controlled by Cygnus) and since our primary goal was Linux, not the compiler itself we decided to work towards an IA-64 backend that generates correct, but not necessarily optimal code. As explained previously, the idea here is that it doesn't matter how poor the generated code is initially—as time progresses, the compiler will improve and taking advantage of an improved compiler will simply require re-compiling the existing code with the latest version of the compiler.

Even the limited goal of producing a functional (not optimal) IA-64 backend was quite challenging and took the better part of four months. At the end of this time the longed for “Hello World!” finally materialized on the computer screen. Once this milestone was reached, it was mostly a downhill ride, fixing various bugs here and there until finally the

point was reached where the backend would pass the entire GNU C test suite.

With the compiler in place, it was theoretically possible to simultaneously start working on the kernel and the user-level support (GNU C library). The only question is how to develop and debug user-level software when the kernel doesn't even exist yet? This is a bit like trying to test-drive a car without an engine! Here is where a simulator comes in handy. We took an existing HP-UX based IA-64 simulator and added Linux system call support to it. This simulator works such that it faithfully executes IA-64 instructions in user-level, but whenever a program attempts to perform a system call, the call is redirected to a small system call stub that implements the desired semantics by making appropriate Linux/x86 system calls. For example, the Linux/ia64 `fstat` call can be implemented by calling the Linux/x86 version of `fstat` and doing some 64-bit to 32-bit conversions to account for differences in the `stat` structure.

With the toolchain and simulator ready to go, we started working on the Linux kernel. While, at this point, we cannot disclose any technical details of this effort, we can say that work has been progressing smoothly. Shortly before Christmas of 1998, enough of the kernel was in place to boot all the way through to the point where the root filesystem would normally be mounted. By the end of February, user-level processes worked reliably. This meant that all the machinery needed for `fork` and `execve` was in place and working properly. A screen dump of a typical boot is shown in Figure 2. The figure shows a literal copy of the boot screen, except that some output that could reveal IA-64 features has been removed and the BogoMIPS number has been replaced by question marks. The reason for the latter step is to avoid potential confusion: the boot screen has been obtained with a (fast) simulator that doesn't

simulate the micro-architectural features of the CPU and hence the BogoMIPS number printed is, well, completely bogus. Thus, rather than risk confusing readers, we opted to remove the reported BogoMIPS number completely. Another artifact of using the simulator is that no real devices exist in the system. Thus, instead of a real serial console, we implemented a `simserial` driver that simulates a console through an xterm window. Similarly, the SCSI disk is actually a regular file on the host's filesystem. This file is accessed through our own `simscsi` driver with the effect that the IA-64 kernel thinks that it's dealing with a normal SCSI device. This is handy because it lets us test the entire SCSI-path without actually requiring a physical device. Though the implementation is completely different, this is not unlike what the `loop` driver does for the normal Linux kernel. In fact, we use the latter driver to access the simulated disk from the host OS which is convenient for bootstrapping and debugging purposes.

4.1 Summary

At this point, it may be helpful to give a brief summary of Linux/ia64 architecture. While details will continue to change as the project progresses, the fundamental architectural features mentioned below are expected to remain fairly stable (but see comments below):

Host platform:	Linux/x86, HP-UX, native
Progr. model:	LP64
Endianness:	Little
Object format:	ELF64/IA-64
Page size:	8KB

First, note that the native support doesn't exist yet. It will be added as soon as the first actual hardware is available. In the meantime, development continues to be hosted on

Linux/x86 primarily, though HP-UX support is available as an alternative.

The user programming model is the standard LP64 model meaning that the C data type “long” as well as pointers are 64-bit in size. This is the same model that has been adopted by all other Linux and UNIX 64-bit platforms in existence and ensures that porting applications to Linux/ia64 will be straight-forward.

For compatibility with x86, the native endian-mode of Linux/ia64 is little-endian. Well-written software is endian-independent, but to ensure even not-so-well written software can be easily ported to Linux/ia64, we decided it is best to stick to little-endian mode.

As far as the object file format is concerned, Linux/ia64 follows the ELF64 standard that has been defined for IA-64 by HP, Intel, and other UNIX vendors.

At present, the Linux/ia64 kernel is configured to use a page size of 8KB. However, care has been taken to define the platform-dependent code of the virtual memory system in such a way that other page sizes can be accommodated by simply changing one or two manifest constants. While changing the page size is potentially painful once a large body of software exists, this approach facilitates experimentation while the kernel is under development and allows us to select the best size once more detailed performance studies are available.

5 Next Steps

In the near term, our plan is of course to continue to work on the Linux kernel and the other components necessary to reach the goal of a self-hosting system. In the longer term, as more of the architecture is being disclosed, we plan to increasingly involve others in the development. Again the goal here is that

Linux/ia64 will be viewed not as something owned by HP, but something that has been developed by the Linux community for the Linux community. Even now we keep the key Linux developers abreast of the Linux/ia64 progress and involve them in major design decisions (to the degree possible) to ensure that we don't stray too far from the “true path” of Linux evolution.

However, no matter how one looks at the Linux/ia64 project, the truth is that the number of people that can work on it will be quite small for the time being. This is not ideal for free software development, but we believe it is still better than not working on Linux/ia64 at all. Moreover, this by no means implies that there is nothing you can do in support of Linux/ia64. Most obviously, making sure that existing and new software is 64-bit clean will help Linux/ia64 deployment a great deal. For example, as of last count, Mozilla still didn't seem to be fully 64-bit clean. Also, if you love hacking compilers, there is lots you can do because many EPIC-style optimizations are quite independent of the nitty-gritty details of an actual architecture. This is further helped by the Trimaran EPIC research suite available from <http://www.trimaran.org/>. This suite contains everything needed to do compiler research with EPIC-style architecture and could be used as a starting point to play with EPIC-optimizations for egcs. Trimaran is freely available in source code for research purposes (though the license does not permit the source code to be used in GPL'd software). If you're interested in pursuing such a project, be sure to check Marc Lehman's web page at <http://www.gcc.ml.org/epic/> as well since this web page has useful background information and is maintained by people interested in adding EPIC-support to egcs.

6 Conclusion

After giving a brief introduction into the IA-64 architecture we described what is involved in bringing Linux to this modern high-performance architecture. Since most of the architecture will remain secret until IA-64 systems are generally available, the Linux/ia64 that was started in February 1998 at HP Labs is quite different from most run-of-the-mill open source projects. In particular, participation in the project requires non-disclosure agreements (NDAs) and the exchange of information, source code, and IA-64 documentation is tightly regulated by HP and Intel. Despite these limitations, the HP Labs project attempts to stay as close to the spirit of an open source project as possible by involving the key developers wherever and whenever appropriate. By doing so, the project has made good progress and at the time of this writing has brought forward a complete GNU-based development toolchain, a Linux/ia64 simulator, and a good start has been made on the Linux kernel itself. Clearly, much remains to be done but we are optimistic on future progress and are looking forward to the day where the source code can finally be released to the rest of the Linux community.

Acknowledgments

I'd like to thank Stephane Eranian (<eranian@hpl.hp.com>) for his contributions to the Linux/ia64 kernel and for writing the user-level utilities that are being used for kernel testing.

References

[1] John Crawford and Jerry Huck. Motivations and design approach for the IA-64

64-bit instruction set architecture. In *Microprocessor Forum*, Palo Alto, CA, October 1997.

<http://www.hp.com/esy/technology/ia.64/products/slides/>.

[2] Carole Dulong. The IA-64 architecture at work. *IEEE Computer*, 31(7):24–32, July 1998.

<http://www.computer.org/computer/co1998/r7024abs.htm>.

[3] Linley Gwennap. Intel outlines high-end roadmap. *Microprocessor Report*, pages 16–19, October 1998.

[4] Tom R. Halfhill. Beyond Pentium II. *BYTE Magazin*, pages ??–??, December 1997.

<http://www.byte.com/art/9712/sec5/art1.htm>.

[5] John L. Hennessey and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., Palo Alto, 1990.

[6] David Mosberger. Linux/Alpha or how to make your applications fly. In *Proceedings of the Third Annual Linux Expo*, Raleigh, N.C., March 1997.

<http://www.mostang.com/~davidm/papers/expo97/paper/>.

```
Linux version 2.2.2 (davidm@hpl.hp.com) (gcc version egcs-
2.91.57 19980901 (egcs-1.1 release)) #61 Mon Mar 1 23:28:59 PST 1999
Calibrating delay loop... ??? BogoMIPS
Memory: 14856k/18432k available (960k code, 2080k re-
served, 504k data, 32k init)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.2
Based upon Swansea University Computer Society NET3.039
Starting kswapd v 1.5
SimSerial driver version 0.3 with no serial options enabled
scsi0 : simulated SCSI host adapter
scsi : 1 host.
  Vendor: HP          Model: SIMULATED DISK      Rev: 0.00
  Type:   Direct-Access          ANSI SCSI revision: 02
Detected scsi disk sda at scsi0, channel 0, id 0, lun 0
scsi : detected 1 SCSI disks total.
SCSI device sda: hdwr sector= 512 bytes. Sectors= 2097152 [1024 MB] [1.0 GB]
Partition check:
  sda: sdal
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 32k freed
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended
root remounted read-write
/proc mounted
Welcome to tsh (TinyShell) version v0.3 (Mar  1 1999, 23:20:18)
/ # /bin/ls -lia
job [5]
  2 drwxr-xr-x  12 0    0          1024 Feb 24 01:08 .
  2 drwxr-xr-x  12 0    0          1024 Feb 24 01:08 ..
 11 drwxr-xr-x   2 0    0        12288 Jan 08 02:31 lost+found
2049 drwxr-xr-x   2 0    0          9216 Jan 30 07:55 dev
2050 drwxr-xr-x   2 0    0          1024 Feb 26 18:48 bin
2051 drwxr-xr-x   2 0    0          1024 Jan 08 03:21 sbin
2052 drwxr-xr-x   3 0    0          1024 Feb 03 19:10 etc
   1 dr-xr-xr-x  11 0    0              0 Mar 02 07:29 proc
  13 drwxr-xr-t   2 0    0          1024 Mar 02 07:22 tmp
  19 drwxr-xr-x   2 0    0          1024 Feb 01 07:44 mnt
  21 drwxr-xr-x  13 0    0          1024 Feb 24 01:06 usr
  35 drwxr-xr-x  13 0    0          1024 Feb 24 01:07 var
job [5] done
/ #
```

Figure 2: Screendump of Linux/ia64 boot