Itanium®– a viable next-generation technology for crash simulation?

Author:

Adrian Hillcoat
Hewlett-Packard Ltd
Bracknell, UK

Correspondence:

Adrian Hillcoat
Hewlett-Packard Ltd
Cain Rd
Bracknell
RG12 1HN
United Kingdom
Tel: +44-(0)1344 365205

e-mail: adrian.hillcoat@hp.com

Keywords:
Itanium®, Parallel Processing, Processor Architecture, System Architecture, Superlinear

ABSTRACT

In the area of crash simulation, LS-DYNA has traditionally been a good application for measuring performance of leading computer architectures. This paper considers the architectural design of the Intel Itanium® processor, as well as the system architectures into which Itanium fits, with the purpose of understanding the benefits of the novel 64-bit EPIC architecture over and above conventional 64-bit RISC architectures as well as 32-bit Intel XEON processor architectures. The paper will provide details of LS-DYNA performance results achieved using distributed memory parallel execution. It will conclude with a look into the future to predict what might be achievable with further generations of this processor architecture for crash codes.

INTRODUCTION

Crash simulation is one of the category of applications known as “high-performance technical computing”, which is to say that it requires state-of-the-art computing facilities in order to achieve industry-competitive results. In an automotive and aerospace world where time-to-market as well as product quality are paramount, then no design group can afford to be behind on this technology curve in order to remain competitive, not to mention conformant to industry safety standards.

LS-DYNA is of course an explicit finite element program for structural mechanics, which means that its code style is rather different than an implicit code in that it does not solve in-core matrices, but rather takes a time-step approach to solving the equations. This paper will investigate how the organization of the code in the program, and its style, is suitable for an architecture such as Intel Itanium®.

Hewlett-Packard co-developed the Itanium® architecture in conjunction with Intel in order to provide a viable high-performance processor to meet the requirements not only of commercial applications, but also of high-performance technical applications such as LS-DYNA. With the IT industry dynamics currently being experienced, it is a contention that ultimately most high-performance systems will be built from commodity components, which is a significant change from the previous two decades where the role of specialist high-performance systems was clear.

The paper will also consider how the advent of 64-bit memory addressing is relevant and useful, as well as the impact of running crash simulations using 64-bit floating-point precision rather than 32-bit. Some interesting and unusual effects of running parallel will also be discussed.

WHAT IS 64-BIT COMPUTING?

It is important at this stage to be very clear about 64-bit computing, and to define some terms. The term ”64-bit” is used in two different ways, which mean completely different things. Firstly, 64-bit floating-point refers to the space taken by a floating-point variable in a processor or memory. It differs from a 32-bit floating-point value in that it has significantly greater accuracy, i.e. 15 decimal places compared with 6 or 7 decimal places, and also a much larger possible range. The range of a number refers to its absolute value. 64-bit floating-point variables are always used in applications where there is either natural “accuracy” inherent in the problem which must be maintained throughout the calculation, or where a calculation is done which involves numbers with widely-differing exponents, and thus for example the order of summation of a series of variables is important. Both of these situations are found in parts of LS-DYNA, and therefore 64-bit floating-point is used accordingly.

However, this is nothing new. It has been possible since the early 1980’s (or even earlier in some architectures) to process 64-bit floating-point, albeit with the potential for running more slowly than in 32-bit floating-point due to the longer execution time for a more complex instruction, allied to the fact that cache and memory utilization will be one half as good as if 32-bit floating-point were used, simply because each variable takes up twice as much space, and thus the available cache or memory is effectively halved.

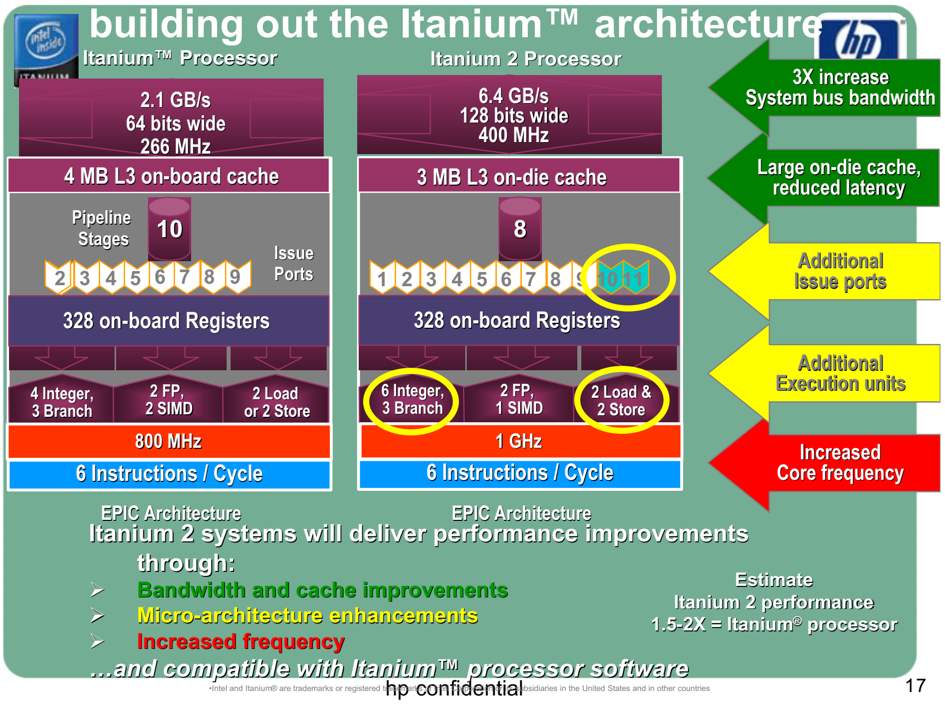
Secondly, the term “64-bit” is also used to refer to how much memory can be addressed by a processor instruction, and it is in this context that the phrase is most often used in the industry today. There are a number of 64-bit processor architectures available today, including Intel Itanium®, IBM POWER4, Sun UltraSPARC®, HP PA-RISC, HP Alpha, etc. The significance of this capability is that it allows for larger memory jobs to be run than with 32-bit architectures, which have a memory-addressing limit of 2GB-3GB, depending on the specific implementation. Some datasets in other high-performance technical computing applications exceed 2GB very easily, and it is these that have adopted the use of 64-bit addressing first, for example Computational Fluid Dynamics, where multi-million cell models require as much as 64GB of memory to compute. This tendency will also only increase, as finer meshes are required for better results.

One peculiar feature of parallel processing is that, notwithstanding the need to address large memory for grid/mesh generation for a “large” problem, if the job can be sub-divided into enough pieces, then each “piece” may well fall below a 2GB or 3GB memory addressing limit. So-called massively-parallel problems illustrate this fact well. For example, a problem that requires 128GB of physical memory to solve could in fact be solved by applying 64 systems to it, each with access to 2GB of memory (assuming rather unrealistically that there are no other overheads!) and therefore a 64-bit architecture is not required (for the solution at least).

Intel’s Itanium® architecture is new in the sense that not only is it a brand-new architecture, unlike any other (in recent years) but that it is also a 64-bit addressing architecture, unlike the IA-32 architecture which can certainly process 64-bit floating-point, but cannot address more than 2-3GB of memory from a single process. 64-bit addressing allows the potential to address (realistically) many Terabytes of memory, certainly far more than any system can deliver today.

ITANIUM® ARCHITECTURE

The Itanium® architecture has been developed as a result of a close co-operation between Intel Corporation and Hewlett-Packard, over a period of around nine years. Its actual architecture is named **E**xplicitly **P**arallel **I**nstruction **C**omputing, or **EPIC**. It differs from current **RISC** (**R**educed **I**nstruction **S**et **C**omputing) and **CISC** (**C**omplex **I**nstruction **S**et **C**omputing) architectures significantly, and is intended to address the shortfalls of these existing architectures.



Itanium 2 systems will deliver performance improvements through:

- Bandwidth and cache improvements
- Micro-architecture enhancements
- Increased frequency

...and compatible with Itanium™ processor software

The shortfalls are perceived as arising from an inability to use all of the computational units in a RISC architecture, all the time. Of course, there are very few real-world applications which could use many or all of the possible instructions, but in fact LS-DYNA, due to its code design, can in fact use many instructions. An example is a multiple sum, where several add units can be used concurrently with data independence. As will be explained later in this paper, this is a key attribute and explains the "conundrum" that a lower speed Itanium® processor can equal or better a much higher clock-speed IA-32 processor in real application performance – an indication that the Itanium architecture is achieving more real work per clock cycle.

To elaborate a little further on this principle, consider an architecture where one instruction is executed each machine clock cycle. If, for argument's sake, this clock cycle is 10ns, then this equates to a "performance" of 100 million instructions per second. In fact, it is hard to conceive of any application or algorithm that could not use this notional instruction each clock cycle. But in fact, this is the crux of the problem, because in real life, there are instructions that actually create something in terms of results, such as adds or multiplies, but there are also instructions that are just as necessary, such as loads and stores, which don't actually create anything, and may have to wait a number of cycles to complete. So, our notional performance of 100 million instructions per second will in fact be reduced to something rather less, even as low as 30 million instructions per second. The real reason for inefficiency in processor architectures is then that it is almost impossible to use all of the power that is there: in our hypothetical case with just one possible instruction issued per machine cycle, it is very unlikely that a real result would be generated each such cycle.

In order to improve on this inefficiency, architectures have been developed in order to allow for "real" work to be done each machine cycle by introducing features such as "pipelining" and out-of-order instructions. The principle of pipelining is easily understood: imagine the start of a shift in a car factory. The process of building a car is rather like the progress of an instruction through a processor, as the end result requires many intermediate steps, such as adding the engine, seats, wheels etc. in the car factory. However, once the first new car appears at the end of the production line, the next one appears shortly afterwards, even if the first one took an hour or two to produce. Pipelined processor architectures are like this too: the first result takes a while to produce, but if the pipeline is full of work, then a result will be produced every clock cycle. Despite all these innovative features in a processor design, however, the result is still one that is relatively inefficient because there are always reasons why a pipeline, for example, cannot continually produce results, rather analogous to a machine in the car factory breaking down, and production is halted albeit temporarily.

Out-of-order execution in a processor is another hardware feature that has been added in order to attempt to improve efficiency. Essentially this is a piece of logic that looks at a queue of instructions, and determines if there is an alternative way to execute them which would keep the processor busier, and pipelines fuller, whilst still producing the correct answers. Despite the large development effort for such a feature, in general, this tends only to ever increase performance by around 5%, compared with an architecture that does not have this capability (although pathological cases exist).

To summarize this discussion so far on processor architectures, it should now be clear that:

Net Performance = Processor Frequency * Number of Useful Instructions/cycle

So, in order to increase the net performance, either the processor frequency should be increased, or the number of useful instructions per cycle needs to be increased, or both.

To take the processor frequency increase possibility first, this is clearly something that the industry does on a regular basis. Moore's Law [1] suggests that every 18 months, the density of transistors on a chip can be doubled, due to improvements in either manufacturing capability or design, and this roughly translates to clock speed increase. As an example, the author is creating this document on a laptop with a processor running at 1.8GHz, which is state-of-the-art for laptops today, whereas 4 years ago, the state-of-the art was around 300MHz. So, in 4 years, clock speed has increased actually at a rate rather greater than that predicted by Moore. However, one day this year-on-year increase must eventually slow, and feature size surely cannot be made ever smaller without some other physical limit being reached).

If ultimately processor frequency might be limited, then it is worthwhile investigating what other possibilities exist for increasing the number of useful instructions per cycle. This is where the Intel Itanium® architecture comes in, as it is a radical departure from the conventional RISC architectures of today.

It is not the intent of this paper to go into inordinate detail about the **EPIC** architecture, but rather to point out the key differences in it, which allow the exploitation of the large number of functional units it has, which then allows for greater efficiency.

The first crucial difference is that **EPIC** has a far greater number (328) of physical registers than current RISC processor designs. Registers are used to hold temporary data: the more the better, and the Itanium processor has four times as many registers as conventional RISC architectures.

Secondly, it has a large number of functional units, and these can be programmed so that they are each working on separate or un-related data, as much as possible, to increase the number of useful instructions executed per machine cycle.

Thirdly, it has a large, close, data cache, of 3MB, for local, low-latency access to data that is being processed. The “closer” the cache to the processor, the more quickly data can be loaded into registers ready to process.

Lastly, the responsibility for ensuring efficient use of these functional units and registers is delegated to the compilers, rather than to run-time hardware. This means that decisions can be made to try to run different pieces of code together so as to maximize efficiency. A good example is the following:

```
DO I=1,NLOOP
.
.
IF (A(I).GE.B(I)) THEN
      some code
ELSE
      alternate code
END IF
END DO
```

This is known as a run-time conditional loop, i.e. an IF condition whose result is only known at run-time, because it depends on the actual value of the array, B. In a conventional processor architecture, the branch would possibly be predicted by the hardware, but in most cases the processor would stall if it could not determine in advance which branch to take. The reason this is a problem is that it breaks all the pipelines and thus reduces efficiency.

For the Itanium® architecture, the compiler would generate code for both sides of the `IF` statement, and then the processor would actually execute both sides of the `IF` at run-time, and then figure out which answers are actually valid and only store those away. It can do this because it has enough registers to store all the temporary variables and also because it has enough functional units to do both calculations. The net result is that because pipelines have been preserved, the actual efficiency of the processor has been increased significantly. For those with an interest in other architectures, this is very similar to a “vector merge” operation which would be done on a vector system such as a Cray or NEC.

To conclude this discussion on processor architecture then, the Itanium® chip is able to deliver greater net performance due to these architectural features which permit more than one effective instruction per cycle to be delivered. It should also be pointed out that maturity of compilers is absolutely key here, and the efficiency of such compilers continues to improve.

SYSTEM ARCHITECTURE

As well as processor architecture, codes such as LS-DYNA depend heavily on the actual system architecture that the processors are installed in. LS-DYNA requires a certain amount of memory bandwidth, and as low memory latency as possible.

To take the subject of memory bandwidth first, it is important to understand that LS-DYNA is different from, for example, structural applications such as MSC/NASTRAN. This is because the code does not deal with large in-core matrices, but with rather smaller data structures. Loops in LS-DYNA are often of length 128-512, or similar, and the loops are fairly tight code accessing perhaps 30 arrays of length 128-512. Therefore, and because much of this data can be in-cache, the memory bandwidth available is important, but not the most important architectural feature.

As already explained, memory latency is a key attribute of a system architecture, and this in fact, along with actual processor performance, drives the real application performance achievable. To illustrate this, the example of earlier HP Unix™ servers will be used. At one point in time, HP had an older high-end Unix™ server known as the V-Class, and a newer mid-range server known as the N-Class. Both systems shipped with the same PA-8500 processor, running at 550MHz. One would therefore expect, all things being equal, that similar performance for LS-DYNA would be achieved. In fact, the N-Class was around twice as fast as the V-Class for LS-DYNA, and this was solely due to the memory latency of the different systems.

To be specific, the V-Class had a memory latency of around 500ns, compared with around 290ns in the N-Class. This latency actually refers to the time taken to load a cache-line of data from main memory into the processor’s registers. Not only was the N-Class latency much lower (290ns compared with 500ns), but the cache-line size in the N-Class was 64 bytes compared with 32 bytes in the V-Class. The reason for the V-Class latency being 500ns is that it has a complex 8x8 memory crossbar architecture, whereas the N-Class has a simpler memory bus architecture. The more complex the memory system, the higher the latency. This is why HP’s newer high-end systems are cell-based architectures, with less-complex 4x4 crossbars, and resulting lower latency.

This phenomenon is rare, but this is a good example of how system architecture can drive the performance achievable on an application, as in this case, the processor was identical. Needless to say, the N-Class was a very successful high-performance technical computing server (as is its cell-based successor, the RP7410).

BENCHMARKS

In order to demonstrate many of the points made so far, some benchmark results for LS-DYNA are now given, along with an analysis. The intent is to show the relatively high efficiency of the Itanium® architecture, compared with the conventional IA32 architecture. This section only considers performance and not price/performance, which of course is also important.

The results that are shown in the following table are from execution of LS-DYNA MPP 960 when run on a cluster of 8 2-way processing nodes, using a simple Gigabit Ethernet interconnect. One cluster comprised HP ProLiant DL360 nodes, with a 533MHz Front Side Bus and 2.8GHz Intel Xeon processors, and running RedHat Linux. The other cluster comprised HP RX2600 nodes with 900MHz Intel Itanium 2 processors, running HP-UX 11i.

What is clear from the results shown is that the results are actually very similar, both in terms of scaling and also actual run-time results. The LS-DYNA benchmark was run on 4, 8 and 16 processors, using the MPP version. Clearly then, and bearing out the theory already presented, the efficiency of the Itanium 2 processor is higher than the Xeon processor, as although the run-time results are similar, the processor clock frequencies are a factor of 3 different (900MHz to 2.8GHz). Other factors can also affect the performance of each system, such as memory speed, memory latency, cache size etc, but it is clear that to some extent, the processor efficiency is very different between the two architectures.

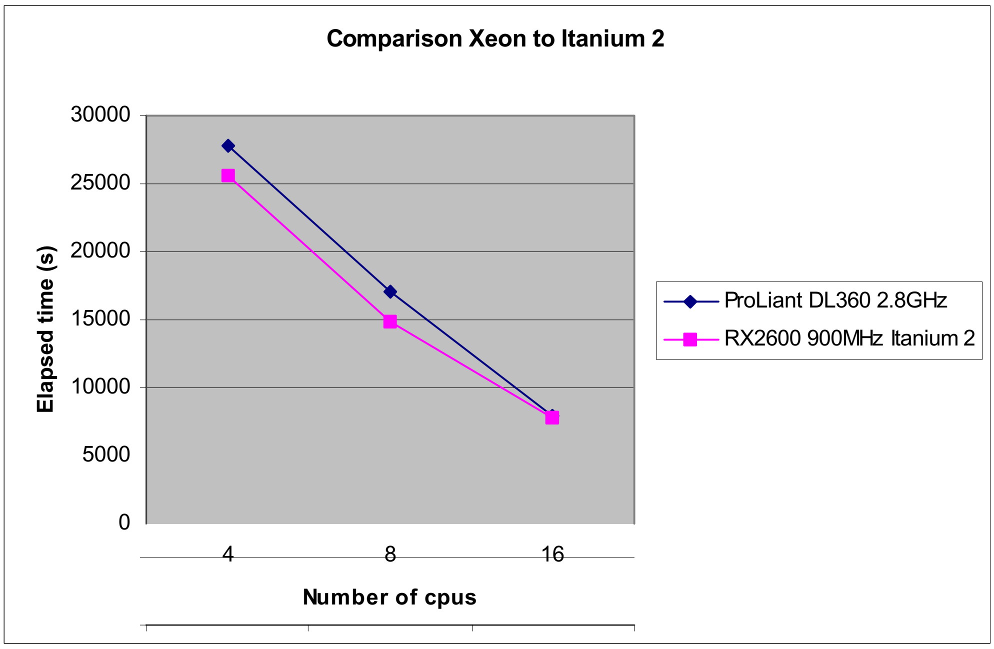


Figure 1 Example of parallel scaling on Xeon and Itanium 2

SUPERLINEAR SPEEDUP

Furthermore, there is a particular phenomenon known as “Superlinear Speedup”. This occurs occasionally as a result of running a LS-DYNA job in parallel, either on shared-memory or distributed-memory systems. Essentially what happens is that at a certain processor count, the combined data cache size becomes large enough to accommodate all of the critical data in the DYNA calculation. For example, a parallel job running on 32 processors would have access to 32 x 3MB of data cache, and it could well be that 96MB of cache is enough to hold most of this critical data, whereas, for example, 90MB might not be enough. At this point, the parallel speed-up, or efficiency, is far greater than at a lower processor count, and therefore the curve showing speed-up against number of processors moves up and to the right. Beyond this specific point, the curve will flatten again, but it does explain why sometimes greater speed-up is achieved than could be expected.

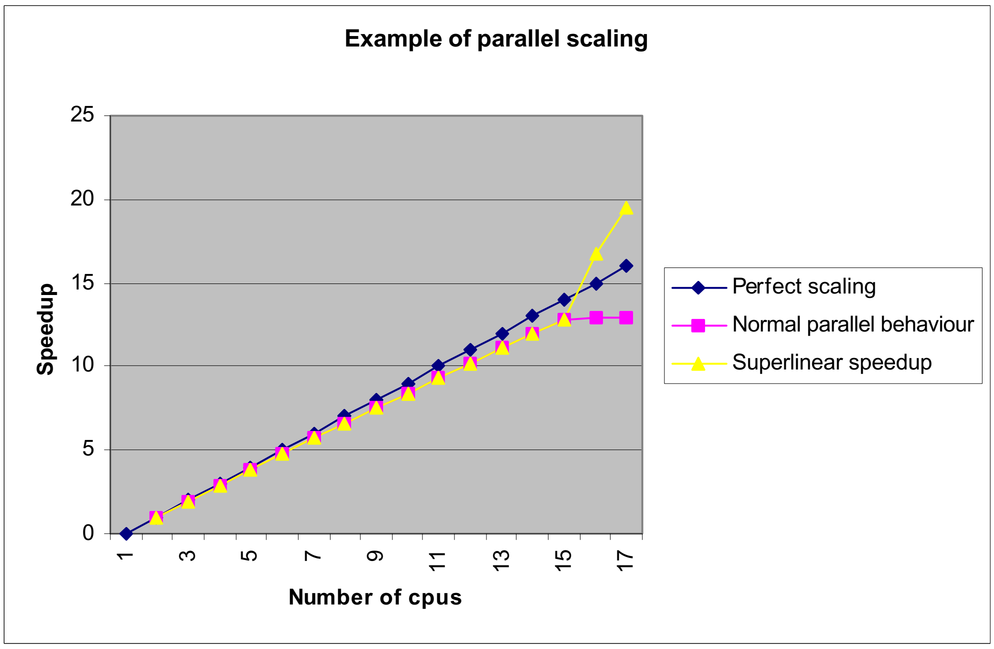


Figure 2 Example of parallel scaling and Superlinear speedup

This phenomenon is really in direct contradiction of Amdahl’s Law [2], which in its simplest form, indicates that the parallel speed-up for an application can never be greater than the number of processors used for the calculation, and usually far less due to scalar code and other overheads.

The importance of data cache memory therefore cannot be understated: the more the better for an implicit code. For applications which solve very large matrices, the benefits of large data cache are less obvious, as for a large enough problem, there could never be enough cache memory with today’s technology to hold all the relevant matrices, and thus data accesses are satisfied from main memory. Main memory accesses are of the order of 10 to 100 times greater latency, and although pipelining of requests can reduce the effects of the increased latency, nevertheless it would always be preferable to access data in cache.

CONCLUSIONS

The conclusion reached is that the Itanium processor technology is able to deliver greater efficiency than a conventional processor on a code such as LS-DYNA. The measure for this is actual performance achieved, compared with the clock speed of the processor. Not only does Itanium show greater performance per clock cycle than conventional RISC architectures, but also it demonstrates greater performance per clock cycle than Intel IA-32 processors. To be specific, a 1GHz Itanium processor is usually equal or better in performance to a 2.8 GHz IA-32 processor.

The future will bring a number of higher-speed, larger-cache Itanium® processors, and Intel’s roadmap is compelling. Already in the public domain is the fact that in mid-2003, Intel will introduce the Madison processor, which will have double the data cache size of the current Itanium 2 processor (6MB compared with 3MB), allied with a very significant increase in clock speed to around 1.5GHz. Follow-on generations of Itanium® will of course continue this exciting trend. It would not be unreasonable to expect to double the existing performance within two years

However, the title of this paper refers to “viable” and as this technology continues to be developed, and its costs to end-users come down, as is inevitable, it is expected that this Itanium® technology will be amongst the price-performance leaders for very demanding applications such as LS-DYNA. Price-performance is the key metric for most users, as there is always a need for performance, but not at any price.

REFERENCES

1. GORDON E. MOORE. (1965)
"Cramming more components onto integrated circuits", Intel Corporation

ftp://download.intel.com/research/silicon/moorespaper.pdf

2. GENE AMDAHL "Amdahl's Law (1967)

For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit co-operative solution...The nature of this overhead (in parallelism) appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate, even if the housekeeping were done in a separate processor...At any point in time it is difficult to foresee how the previous bottlenecks in a sequential computer will be effectively overcome

http://home.wlu.edu/~whaleyt/classes/parallel/topics/amdahl.html

3. EPIC TECHNOLOGY MOVES FORWARD

Intel Corporation, © 2002
http://www.intel.com/ebusiness/pdf/prod/itanium/wp022404.pdf