Crusoe: an Approach for the New Era Computing

Vasco Nuno Barreiro Capitão Miranda

Departamento de Informática,Universidade do Minho
4710 Braga, PORTUGAL
vasco@3ware.pt

Abstract – In a era in where mobile computing becomes imperative, this communication looks for solutions that may turn it technically available. In this context, the capacity of integration and its behaviour with overheating are the main goals or purposes. This communication makes an incursion into the technology behind this new Crusoe processor, which attempts to solve some of these problems and sets new standards for processors design.

1 Introduction

In January of 2000, Transmeta Corporation introduced the Crusoe processors, an x86-compatible family of solutions that combines performance with low power consumption. As might be expected, a new technology for designing and implementing microprocessors underlies the development of these products. As might not be expected, the new technology is fundamentally software based. The power savings come from replacing large numbers of transistors with software

Crusoe is designed for the mobile market, namely for notebooks, webpads and Personal Digital Assistant (PDA). It is not designed to compete in the desktop and server market. Unlike with IDT (Integrated Device Technology, Inc.) and Rise (Rise Technology Company) this is not so much out of necessity, but also the intention from the start. Rise and IDT created CPU's which were simply not powerful enough to compete with AMD and Intel. Their CPU's were also not cheap enough to make a difference, so the only place left for them was the mobile-market. Unfortunately AMD's and Intel's own mobile designs were just as cheap and were also almost as efficient in power-usage. [3]

The Crusoe processor solutions consist of a hardware engine logically surrounded by a software layer. The engine is a very long instruction word (VLIW) CPU capable of executing up to four operations in each clock cycle.[4] The VLIW's native instruction set bears no resemblance to the x86 instruction set; it has been designed purely for fast low-power implementation using conventional CMOS fabrication. The surrounding software layer gives x86 programs the impression that they are running on x86 hardware.

The software layer is called Code Morphing software because it dynamically "morphs" x86 instructions into VLIW instructions. The Code Morphing software includes a number of advanced features to achieve good system-level performance. The Transmeta designers have rendered some functions in hardware and some in software, according to the product design goals and constraints. Different goals and constraints in future products may result in different hardware-software partitioning.

Transmeta's Code Morphing technology changes the entire approach to designing microprocessors. By demonstrating that practical microprocessors can be implemented as hardware-software hybrids. It has dramatically expanded the design space that microprocessor designers can explore for optimum solutions.

2 Crusoe Processor Fundamentals

With the Code Morphing software handling x86 compatibility, has been created a very simple VLIW engine with two integer units, a floating point unit, a memory (load/store) unit, and a branch unit. A Crusoe processor long instruction word, called a molecule, can be 64 bits or 128 bits long and contain up to four RISC-like instructions, called atoms. All atoms within a molecule are executed in parallel, and the molecule format directly determines how atoms get routed to functional units; this greatly simplifies the decode and dispatch hardware. Figure 1 shows a sample 128-bit molecule and the straightforward mapping from atom slots to functional units. Molecules are executed in order, so there is no complex out-of-order hardware. To keep the processor running at full speed, molecules are packed as fully as possible with atoms. [4]

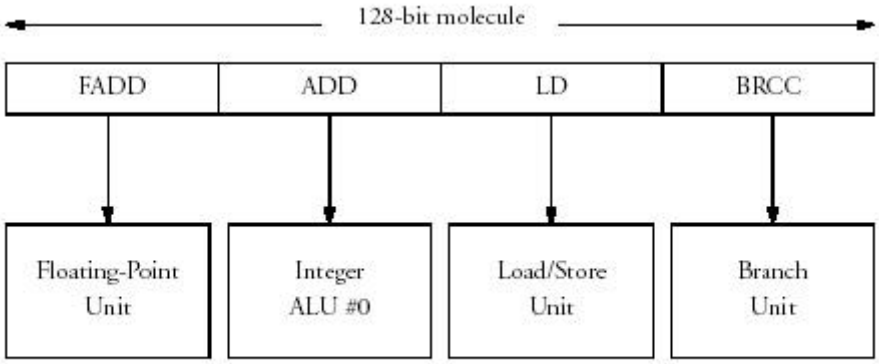


Figure 1. A molecule can contain up to four atoms, which are executed in parallel.

Superscalar out-of-order x86 processors, such as the Pentium II and Pentium III processors, also have multiple functional units that can execute RISC-like operations (micro-ops) in parallel. Figure 2 depicts the hardware these designs use to translate x86 instructions into micro-ops and schedule (dispatch) the micro-ops to make best use of the functional units. Since the dispatch unit reorders the micro-ops as required to keep the functional units busy, a separate piece of hardware, the in-order retire unit, is needed to effectively reconstruct the order of the original x86 instructions, and ensure that they take effect in proper order. This type of processor hardware is much more complex than the Crusoe processor's simple VLIW engine.

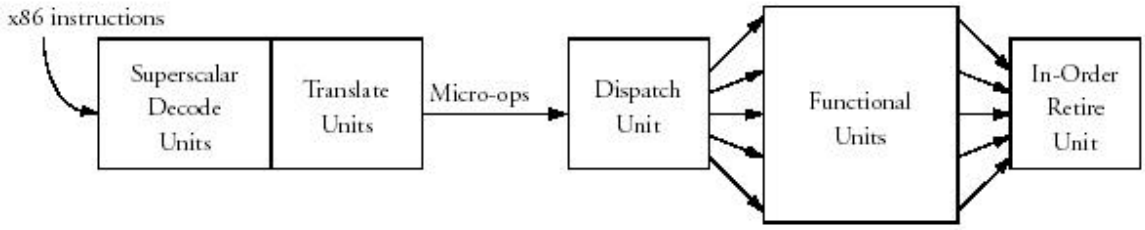


Figure 2. Conventional superscalar out-of-order CPUs use hardware to create and dispatch micro-ops that can execute in parallel.

Because the x86 instruction set is quite complex, the decoding and dispatching hardware requires large quantities of power-hungry logic transistors; the chip dissipates heat in rough proportion to their numbers. Table 1 compares the sizes of Intel mobile and Crusoe processor models.

	Mobile PII	Mobile PII	Mobile PIII	TM3120	TM5400
Process	.25m	.25m shrink	.18m	.22m	.18m
On-chip L1 Cache	32KB	32KB	32KB	96KB	128KB
On-chip L2 Cache	0	256KB	256KB	0	256KB
Die Size	130mm^2	180mm^2	106mm^2	77mm^2	73mm^2

Table 1. The Code Morphing software simplifies chip hardware.

Viewing power dissipation as heat, Figure 3 and Figure 4 contrast the operating temperatures of a Pentium III and a Crusoe processor running a software DVD (Digital Versatile Disk) player. The model TM5400 requires no active cooling, whereas the Pentium III processor can heat to the point of failure if it is not aggressively cooled. [1]

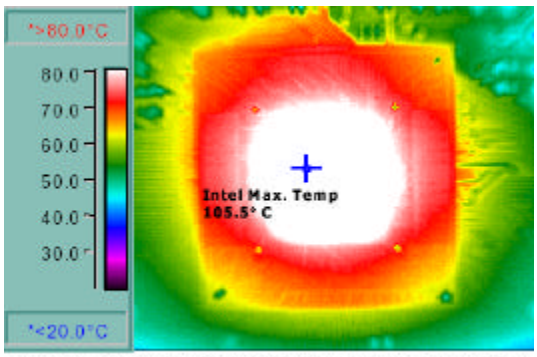


Figure 3. A Pentium III processor plays a DVD at 105° C (221° F).

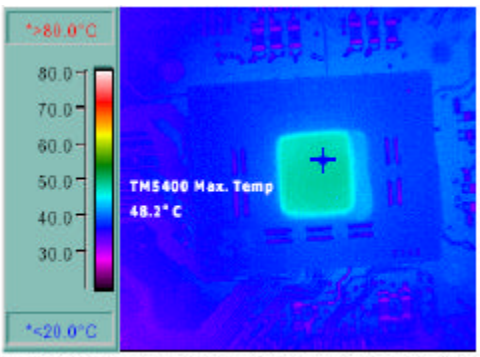


Figure 4. A Crusoe processor model TM5400 plays a DVD at 48° C (118° F).

3 Technology Perspective

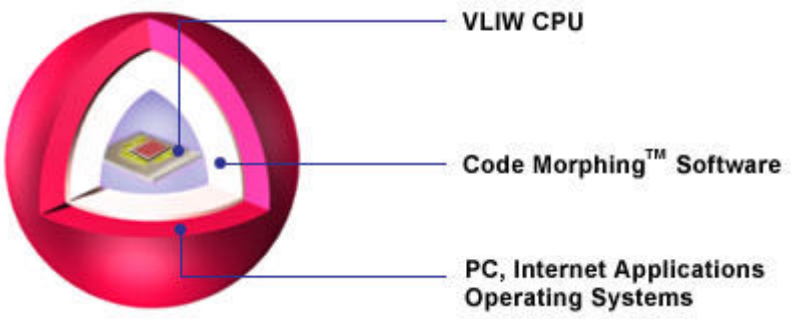
The underlying hardware can be changed radically without affecting legacy x86 software. Each new CPU design only requires a new version of the Code Morphing software to translate x86 instructions to the new CPU's native instruction set.

For the initial Transmeta products, models TM3120 and TM5400, the hardware designers opted for minimal space and power. By eliminating roughly three quarters of the logic transistors that would be required for an all-hardware design of similar performance, the designers have likewise reduced power requirements and die size. However, future hardware designs can emphasize different factors and accordingly use different implementation techniques.[4]

4 Code Morfing Software

The Code Morphing software is fundamentally a dynamic translation system; a program that compiles instructions for one instruction-set architecture into its own instruction set. The entire Code Morphing software is implemented in flash ROM and is the first program to start on boot. Therefore the Crusoe processor, running its Code Morphing software, is

indistinguishable from a 'normal' x86 processor, as the only thing any x86 code sees is the Code Morphing software, acting and operating just like any other x86 processor. The only program written especially for the Crusoe's VLIW engine is the Code Morphing software itself.[2]



The native instruction set can be changed arbitrarily without affecting any x86 software at all. The only program that needs to be changed is the Code Morphing software. One other big advantage is that it solves a problem that has hampered acceptance of VLIW processors. A traditional VLIW processor exposes details of the processor pipeline to the compiler; as a result, any change to that pipeline would require all existing binaries to be re-compiled to make them compatible with the changed pipeline. In other words, any modification to the processor's hardware would require all programs that run on it to be re-compiled.

The difficulties of changing a processor's hardware; like MMX, SSE and 3DNow! instructions, and all hardware implementations for which the software has to be re-compiled. This, however, is not a problem with the Crusoe processor, since, in effect, the Code Morphing software always transparently 're-compiles' the x86 code it is running.

This translation of one instruction set's instructions into another's comes at a price, though; the processor has to dedicate some of its processing power to running Code Morphing, which a conventional x86 processor could have used to execute application code. And although Transmeta has undoubtedly designed the Code Morphing software for maximum efficiency and low overhead, it will never perform on par with a x86 CPU of the same configuration and clockspeed, simply because it has to execute the Code Morphing software before it can actually process an x86 instruction.

4.1 Execution, Decoding and Scheduling

Current superscalar x86 processors, such as the Intel Pentium III, also have multiple functional units that can execute RISC-like operations in parallel; however due to the out-of-order execution a separate piece of hardware is needed to re-construct the sequence of the original x86 instructions and make sure that they execute in order. As a result, this type of processor is much more complex than the Crusoe processor's VLIW engine.

As mentioned above, due to a conventional x86 processor's out-of-order execution, it requires additional hardware to make sure that x86 instructions are executed in the correct order. [5] In contrast, Code Morphing can translate an entire group of x86 instructions at once, whereas a conventional x86 processor translates each instruction separately. Also, the Code Morphing software translates instructions once and stores the result in a 'transla-

tion cache'; the next time the same code is executed, the processor can immediately run the existing instruction from the translation cache. [4]

This software translation opens up new possibilities, since a conventional out-of-order processor has to re-translate and schedule an instruction each time it executes, very quickly. With Code Morphing, the translation process can be optimized by looking at the generated code and minimizing the number of instructions executed. In other words, Code Morphing can speed up execution plus reduce power consumption. However, performance might not be at its peak after the first iteration of the optimization process. It will probably require multiple passes because the code optimization is done in steps. All code is first re-compiled quickly, but not necessarily efficiently, to keep the program and the processor running. Then, when a section of code is run again, it moves up in the hierarchy and is scheduled for optimization, sections that only occur once usually don't get optimized. As a result, that section might take several passes to get fully optimized.

In essence it is no different than programming in a higher language, for example C++. After we've compiled and executed our program, one of the things we're keen on determining is where the performance bottlenecks are. Then we subsequently use assembly to speed up just those sections, and thus optimize the program's overall execution. Programming in C++ gets our program running faster than programming it all in assembly, but the extra performance gained by using assembly where needed can really speed up processing.

4.2 Caching and Optimisation

Although the Code Morphing software is implemented in ROM, it first gets copied to DRAM at boot up to increase performance, residing in a separate memory space along with the translation cache.

The optimisation of the translation process by the Code Morphing software allows for the translation cache to be used much more efficiently, as a result of which the hardware can then execute the optimised translation at full speed. Furthermore, as a part of this optimization, when an application executes Code Morphing 'learns' more about the program and constantly optimizes and speeds up execution.

The Code Morphing software has many ways to gather feedback about a running program, one of which is the 'instrument' translation; during the translation, code is added whose sole purpose is to collect information about the block to be executed. This data is used later to decide when and what to optimize and translate.

A very good example, already mentioned, is knowing how often a piece of x86 code is executed and if often, then optimise for that code, instead of for code used only once or twice, where such an optimisation is a waste of time.

However, the Code Morphing software runs off of main memory and part of the processor's performance will be determined by the bandwidth of the memory interface and the memory technology used. Because the data to be processed and Code Morphing software both reside in memory, this has more performance impact than with a regular x86 processor, which only stores data in main memory.

5 Conclusion

The Transmeta is not designed to be a speed-king but to be powerful enough to do the job. A notebook or webpad doesn’t require 60+ fps in UnReal Tournament. It does however offer a significant lower power-usage. The Transmeta chip even uses less power than

AMD’s special embedded K6-2E chip. Pricing is within the same league so they might have a chance at success, especially because of the large number of usage options as a result of the flexible design.

Transmeta's Crusoe processor sets new standards for processor design and in software translation of instruction sets. We have yet to see the implications of this approach for the computer industry, but they are likely to become apparent over the next several years. The technology is scalable and not limited to low-power designs or to x86-compatible processors. However, due to its architecture it may not offer the same level of performance as any given processor of similar clockspeed and configuration.

Performance can vary greatly between programs, depending on whether its speed of execution relies on rerunning a section of code a large number of times. The Code Morphing software is designed to optimize instructions that recur, allowing execution to be optimized after each iteration. It does offer a great basis for running multiple processor platforms on a single processor core.

When notebooks incorporating Transmeta's Crusoe processor become available, don't expect all of our programs to work faster over time; some code is not too well suited for optimisation, after all, and other code is already heavily optimised.

We must keep in mind, that Transmeta's Crusoe processor emulates another processor's instruction set, and we've yet to come across an emulator that outperforms the original. Although marketing slogans and product presentations might have got we thinking otherwise, in reality you will always lose performance if something isn't programmed natively.

References

[1] Daniel McKenna, “Mobile Platform Benchmarks”, Transmeta Corporation White Paper, January 2000.

[2] Alexander Wolfe, “The Software Side of Crusoe”, Embedded Systems Programming Site, 2000, (www.embedded.com/internet/0004/0004ia3.htm)

[3] Sander Sassen, "Transmeta's Crusoe, HotRod or Performance Hog?”, May 11, 2001, (http://www.hardwareanalysis.com/content/editorials/)

[4] Alexander Klaiber, “The Technology Behind Crusoe Processors”, 2000 (www.transmeta.com)

[5] Hwang, K. Advanced Computer Architecture, McGraw-Hill, 1993.