

“The Technology Behind Crusoe™ Processors: Low-Power x86-Compatible Processors Implemented with Code-Morphing™ Software”

Alexander Klaiber
Transmeta Corporation

January 2000

Transmeta
CORPORATION

presented by nick black <nickblack@linux.com> for cs8803dc 2010-04-15
watch this space for valuable addenda -- BIG MONEY! BIG PRIZES! YOU LOVE IT!!

Motivation

- Commercial processor built around binary translation.
 - Anyone remember the M680x0 emulator for PowerPC Macs? (*)
 - How about PRISM's Epicode + Mica? VEST/AEST on Alpha? (**)
- One of two major GP-VLIW implementations.
 - Yes, I absolutely am discounting Multiflow Computer's 125 sales.
- Integrated design of architecture and translator.
- Interesting design space:
 - An attempt to reduce power and size of PC2001/x86.
 - *Not* targeted at embedded space, where cost is a main motivator!

“A Microprogrammed Implementation of an Architecture Simulation Language” (1977)

(*) Tom Hormby's *IBM, Apple, RISC, and the Roots of the PowerPC* and Steven Levy's *Insanely Great*. (**) Paul Bolotoff's *Alpha: The History in Facts and Comments*.

Anti-Motivation



TRANSMETA

**"YOU PITIFUL ISOLATED BANKRUPTS, YOUR ROLE IS PLAYED OUT.
GO THEN WHERE YOU BELONG, TO THE RUBBISH HEAP OF HISTORY! "**

(* 1917-10-23, paraphrased from John Reed's *Ten Days That Shook the World* (1919)

pull over; that table's too fat (woop woop)

■ [TM8300/TM8600 product spec](#)

■ [TM3200 product spec](#)

Family	Model	Interface	Frequencies	L1 Caches	L2 Cache	TLB	Memories	Peripherals
Crusoe (128-bit + MMX)  TM5400	TM3200	BGA-474	366--400MHz	64K 8-associative code, 32K 8-associative data	none	256 entries, 4-associative	SDR SDRAM 66--133MHz	33MHz 32-bit PCI
	TM5400	BGA-474	500--700MHz	64K 8-associative code, 64K 16-associative data	256K 4-associative unified	256 entries, 4-associative	SDR/DDR SDRAM 66--133MHz, DDR SDRAM 100--133MHz	33MHz 32-bit PCI
Efficeon (256-bit + SSE2)  TM8600	TM8300	BGA-783	900MHz--1.1GHz	128K 4-associative code, 64K 8-associative data	512K 4-associative unified	256 entries, 4-associative	100--166MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x
	TM8600	BGA-783	900MHz--1.1GHz	128K 4-associative code, 64K 8-associative data	1M 4-associative unified	256 entries, 4-associative	100--166MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x
	TM8620	BGA-592	900MHz--1.1GHz	128K 4-associative code, 64K 8-associative data	1M 4-associative unified	256 entries, 4-associative	100--200MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x
	TM8800	BGA-783	1.1GHz--1.7GHz	128K 4-associative code, 64K 8-associative data	1M 4-associative unified	256 entries, 4-associative	100--200MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x
	TM8820	BGA-592	1.1GHz--1.7GHz	128K 4-associative code, 64K 8-associative data	1M 4-associative unified	256 entries, 4-associative	100--200MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x

See Also

- The [sandpile](#) has a fine Transmeta doc collection
 - And a [Crusoe](#) reference
 - Not to mention an [Efficeon](#) reference
- A good [BGA page](#) at HubPages.

Categories: [Hardware](#) | [X86](#)

Sources: Transmeta product datasheets, UIUC CS433 "Processor Presentation Series" notes for Transmeta Crusoe, sandpile.org *IA-32 Implementation Guides* for Crusoe/Efficeon

Initial reactions, pre-paper:

- Anyone can run an x86 translator/emulator
 - Why wouldn't Intel just build this instead?
 - P6 was doing hardware CISC-to-RISC (CRISC) in 1995
...though dissipating serious wattage to do so...
- An *ad hoc* taxonomy of binaries:
 - Targeted (source- and compiler-optimized for this model)
 - Native (compiler-optimized for this model)
 - Fat binaries are native for multiple models, but not forward-native
 - Legacy (“optimized” for minimum compatible feature set)
 - Foreign (ISA-incompatible)
- Without an ISA to target, all binaries are foreign!
 - No programs are designed for our architectural tricks
 - Recompilation can't help us, even if we have source
 - Upshot: Transmeta's no better than its binary translator.
 - Transmeta's hardware and translator must beat out native hardware and a compiler. Hardware differentiation is difficult to utilize, and in any case limited to the CPU.

A straw poll!

Can a translator beat a compiler?

Show of hands?

- McManus, a fisher of men, gets a +1 bonus from Charisma
 - Railing gets a conciliatory, but useless, $+\pi/8$ from General Excellence

Zl bcvavba: uvtuyl qhovbhf;
pregnvayl abg n ohfvarff gb trg vagb(*)).

(*) Toggle ROT-13 in Vim with “ggVGg?” (as you might expect)

A straw poll!

Can a translator beat a compiler?

Show of hands?

My opinion: highly dubious;
certainly not a business to get into.

The Claims:

- Avoid X86's decoding frontend for major power savings.
 - But running the Code-Morphing™ Software means more time working, and thus less time powered down...
 - Intel introduced an MSR0M for low-power provision of complicated instructions in the Core™ microarchitecture.
- In-order VLIW can compete with out-of-order CRISC.
 - Large instruction caches and 64/128-bit operation
 - Remember, VLIW != EPIC and Transmeta != Itanium!
 - Surprising! The translator almost certainly does dependency analysis (it's effectively replacing hardware OOO) – you oughtn't need source to annotate.
- Real power, size and heat savings are effected.
 - Size: Half the die of a “mobile PII” (but 70% of “mobile PIII”)
 - LongRun™ Power Management: 0.4W to 2.2W(!) maximum.
 - Full implementation of (then-nascent) ACPI C-, V- and P-states
 - Intel had to wait for Pentium®-M's SpeedStep®, AMD's PowerNow!™(*)

(*) See “Analysis of Thermal Monitor features of the Intel® Pentium® M Processor” and “Energy-efficient Processor Design Using Multiple Domains with Dynamic V&F Scaling.”

When Klaiber uses quotes, he lies:

“COINCIDENTALLY, hiding the chip’s ISA behind a software layer also avoids a problem that has in the past hampered the acceptance of VLIW machines. A traditional VLIW exposes details of the processor pipeline to the compiler, hence any change to that pipeline would require all existing binaries to be recompiled to make them run on the new hardware. Note that even traditional x86 processors suffer from a related problem: while old applications will run correctly on a new processor, they usually need to be recompiled to take full advantage of the new processor implementation. This is not a problem on Crusoe processors, since in effect, the Code-Morphing™ software always transparently “recompiles” and optimizes the x86 code it is running.” (page 8)

I'm afraid, Mssr. Klaiber, that it'll take more than quotation marks to make recompilation from binary translation.

Proving the 90/10 rule via...Jungian reference?

“Some benchmark programs attempt to exercise a large set of features in a small amount of time, with little repetition -- a pattern that differs significantly from normal usage. (When was the last time you used every other feature of Microsoft Word exactly once, over a period of a minute?)” (page 9)

...exactly whom is being addressed here?

Beyond that: why would meaningless benchmarks be used?
Why worry about something so clearly inadmissible?

Without nicotine's immediate infusion, I will feed the Student Center ATM its long-craved stray cats.



...so let's reconvene in 5 minutes. In the meantime, meditate upon FORTRAN's Eightfold Path of Virtue.

Thanks!
That hit the spot!

AT FIRST I WAS LIKE



BUT THEN I WAS LIKE



Let us return to Klaiber 1990, aka BULLSHIT PATTY BETWEEN TWO SLICES OF LIES.

Further questionable assertions

“In a second pass, the optimizer applies well-known compiler optimizations to the code, such as common subexpression elimination, loop invariant removal, or dead code elimination (including unnecessary settings of the condition codes).” (page 11)

Nothing's won here; by virtue of being well-known, we can assume such optimizations to have been performed during the original compilation.

“This exemplifies optimizations that a hardware-only x86 implementation cannot do: a software-based translation system can actually eliminate atoms from the instruction stream, rather than just reorder them.” (*ibid*)

Is Klaiber telling a baldfaced lie? Does he suffer grave delusions regarding computability theory? Do I? I'm pretty sure this is nonsense. (*)

(*) Exhibits A and B: “Macro-” and “Micro-fusion”. *Intel 64 and IA-32 Software Optimization Guide*, from the Core™ microarchitecture onwards.

Two points well worth repeating

“Though the molecules are executed in-order by the hardware, they perform the work of the original x86 instructions out of order.” (page 12)

“The molecules explicitly encode (*) the instruction-level parallelism, hence they can be executed by a simple (and hence fast and low-power) VLIW engine; the hardware need not perform any complex instruction reordering itself.” (*ibid*)

Is the work done to perform the translations truly less than the work done to decode instructions in hardware? (**)

(*) Advocates of EPIC (primarily at Intel and HP) may yet beg to differ. (**) By the same 90/10 rule, how does the Loop Stream Detector alter this equation (Core™ and Nehalem)?

What?

“Normal atoms only update the working copy of the register. When execution reaches the end of a translation without encountering an exception, a special commit operation copies all working registers into their corresponding shadow registers, indeed committing the work done in the translation. On the other hand, if any x86-level exception occurs inside the translation, the runtime system undoes the effects of all molecules executed since the start of the translation. This is done via a rollback operation which copies the shadow register values (last committed at the end of the previous translation) back into the working registers.” (page 13)

Does this sound like an ROB to anyone else?

Weren't we trying to avoid an ROB?

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. Clearly, this type of processor hardware is much more complex than the Crusoe processor's simple VLIW engine.

Clearly!

I do not think that word means what you think it means.

“The Crusoe host provides innovative alias hardware that addresses this problem. When the translator moves a load operation ahead of a store operation, it converts the load into a load-and-protect (*) (which in addition to loading data also records the address and size of the data loaded) and the store into a store-under-alias-mask (which checks for protected regions). In the (unlikely) event that the store operation overwrites the previously loaded data, the processor raises an exception and the runtime system can take corrective action. **Using this mechanism, it is always safe to reorder memory loads and stores.**” (page 14)

We can agree that this requires physical addresses?

What happens if the reordered code happens to manage page tables? Is this **always** safe?

Perhaps someone might offer a proof?

“Unlikely” is fair, since most deps are detected by the translator. This is only for truly dynamic deps.

See also Itanium's NaT and NaTVal 1-bit “deferral token” registers for misspeculation (*Intel Itanium® Architecture Software Developer's Reference*).

Inventing memory protection.

“At times, x86 instructions in memory get overwritten, either because the operating system is loading a new program, or because an application is using self-modifying code. When this happens to code that has already been translated, the Code Morphing software needs to be notified to keep it from erroneously executing a translation for the old code. To this end, whenever the system translates a block of x86 code, it write-protects the page of x86 memory containing that code.” (page 14)

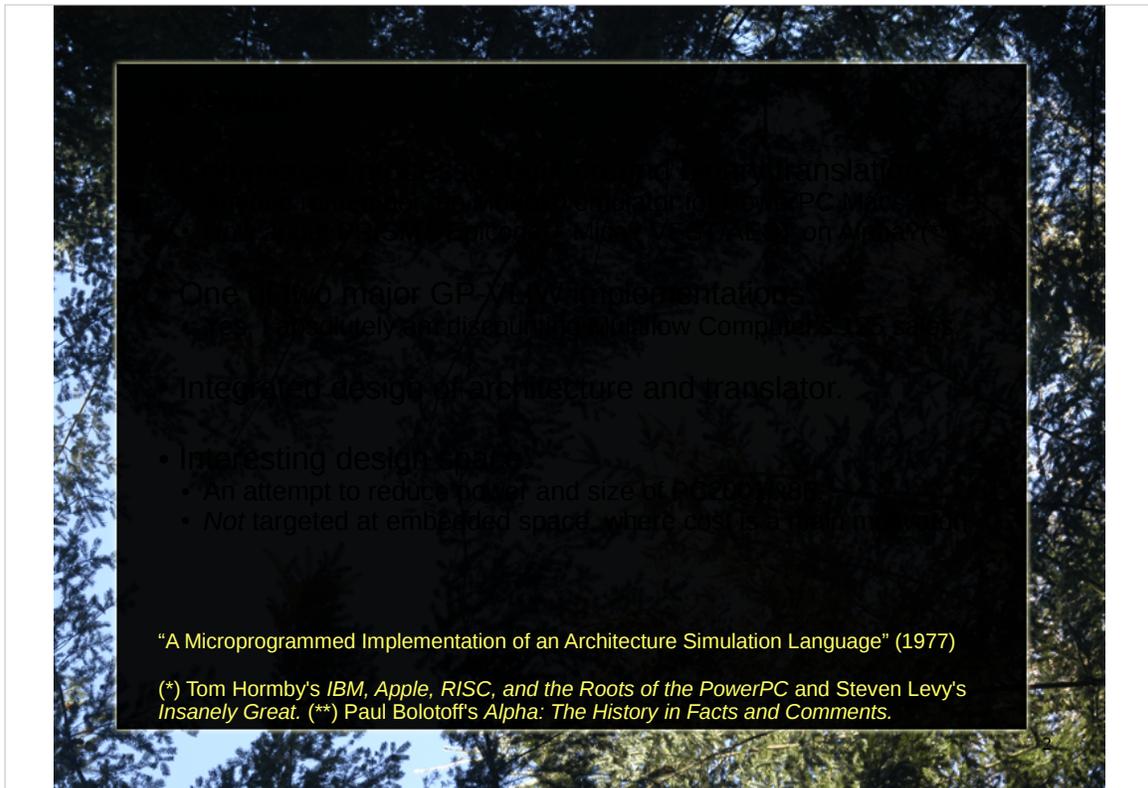
...yes, we mark code read-only, unless (like Xorg's OpenGL or many virtualization systems) we want to write to it. (*)
This is why we have `mprotect(2)` and `mremap(2)`.

Instruction caches have the same issues.

...and, is there really no compilation context preserved across process lifetimes? Have fun, “`find / -exec foo {} \;`” (**)

(*) It was the NX bit which Intel added later; write protection has existed for five presidential administrations. (**) For this and other reasons, we use “`find / -print0 | xargs -0 foo`”, right?

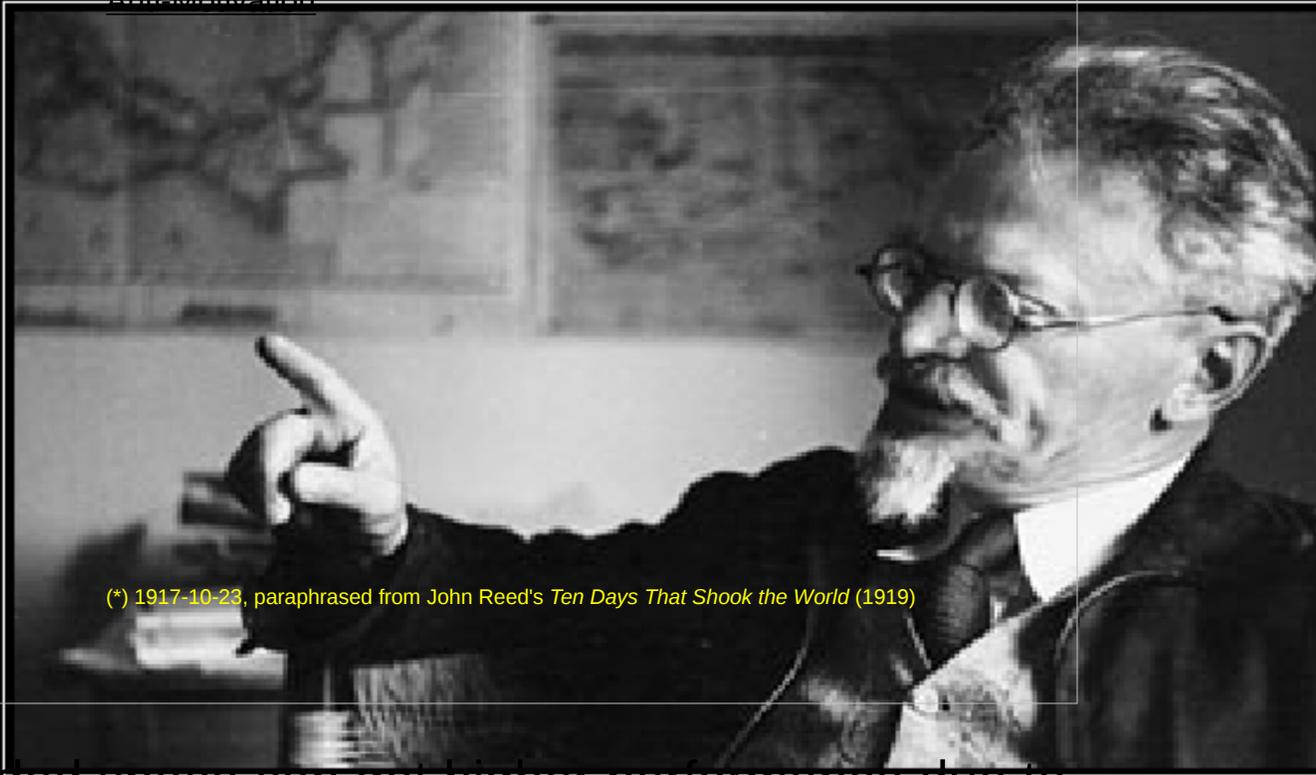




Embedded space can get higher performance due to single-nature approach of chips (especially ASIC/FPGA). We want a general-purpose CPU.

In fact, we want the **most** general-purpose CPU: one designed at its very core to run other CPU's binaries (ie, no planned use of compilers targeting Transmeta)

“Rubbish heap of history”: Trotsky's phrase for the Mensheviks (Russian word for “minority” vs Lenin's Bolsheviks (“majority”))



(* 1917-10-23, paraphrased from John Reed's *Ten Days That Shook the World* (1919)

TRANSMETA

**"YOU PITIFUL ISOLATED BANKRUPTS, YOUR ROLE IS PLAYED OUT.
GO THEN WHERE YOU BELONG, TO THE RUBBISH HEAP OF HISTORY!"**

(ie, no planned use of compilers targeting Transmeta)

“Rubbish heap of history”: Trotsky's phrase for the Mensheviks (Russian word for “minority” vs Lenin's Bolsheviks (“majority”))

[TM8300/TM8600 product spec](#)
[TM3200 product spec](#)

Family	Model	Interface	Frequencies	L1 Caches	L2 Cache	TLB	Memories	Peripherals
Crusoe (128-bit + MMX)  TM5400	TM3200	BGA-474	366-400MHz	64K B-associative code, 32K B-associative data	none	256 entries, 4-associative	SDR SDRAM 66-133MHz	33MHz 32-bit PCI
	TM5400	BGA-474	500-700MHz	64K B-associative code, 64K B-associative data	256K 4-associative unified	256 entries, 4-associative	SDR/DDR SDRAM 66-133MHz, DDR SDRAM 100-133MHz	33MHz 32-bit PCI
Efficeon (256-bit + SSE2)  TM8600	TM8300	BGA-783	900MHz-1.1GHz	128K 4-associative code, 64K B-associative data	512K 4-associative unified	256 entries, 4-associative	100-166MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x
	TM8600	BGA-783	900MHz-1.1GHz	128K 4-associative code, 64K B-associative data	1M 4-associative unified	256 entries, 4-associative	100-166MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x
	TM8620	BGA-592	900MHz-1.1GHz	128K 4-associative code, 64K B-associative data	1M 4-associative unified	256 entries, 4-associative	100-200MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x
	TM8800	BGA-783	1.1GHz-1.7GHz	128K 4-associative code, 64K B-associative data	1M 4-associative unified	256 entries, 4-associative	100-200MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x
	TM8820	BGA-592	1.1GHz-1.7GHz	128K 4-associative code, 64K B-associative data	1M 4-associative unified	256 entries, 4-associative	100-200MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x
	TM8820	BGA-592	1.1GHz-1.7GHz	128K 4-associative code, 64K B-associative data	1M 4-associative unified	256 entries, 4-associative	100-200MHz DDR SDRAM + SPD	400MHz HyperTransport, AGP 1x/2x/4x

See Also

- The [sandpile](#) has a fine Transmeta doc collection
- And a [Crusoe](#) reference
- Not to mention an [Efficeon](#) reference
- A good [BGA page](#) at HubPages.

Categories: Hardware | X86

Sources: Transmeta product datasheets, UIUC CS433 "Processor Presentation Series" notes for Transmeta Crusoe, sandpile.org IA-32 Implementation Guides for Crusoe/Efficeon

A straw poll!

Can a translator beat a compiler?

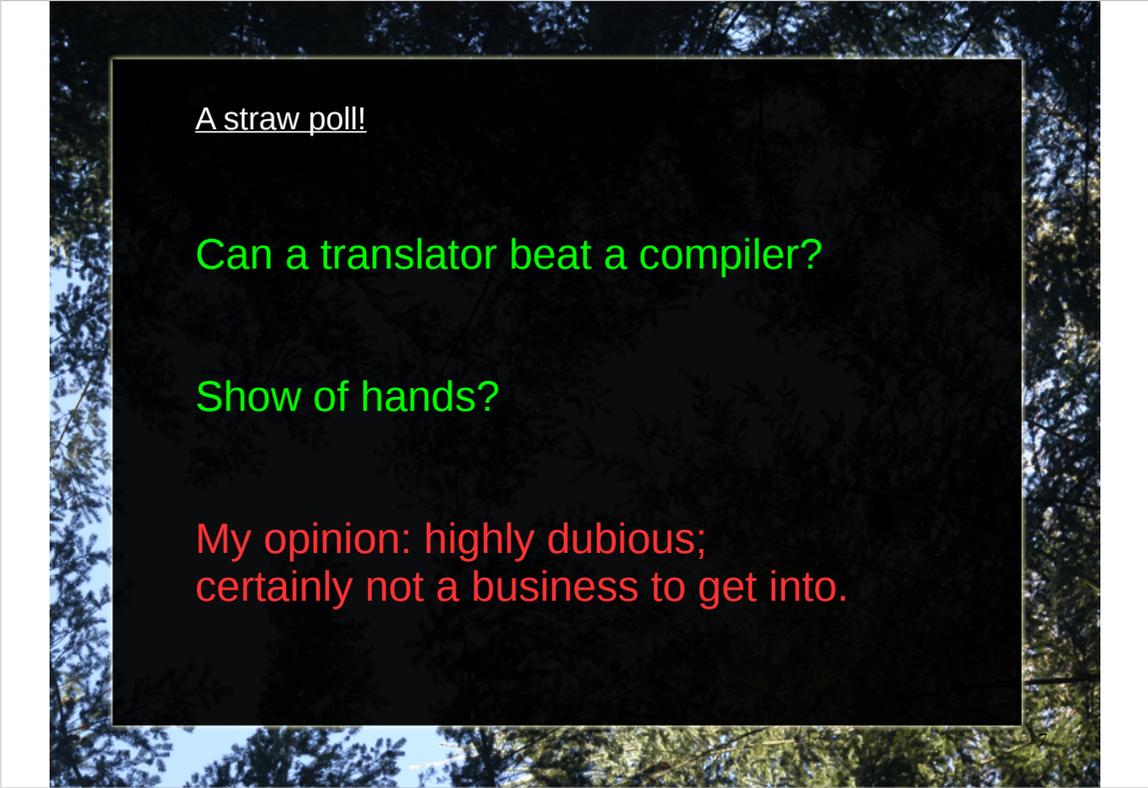
Show of hands?

McManus, a fisher of men, gets a +1 bonus from Charisma

Railing gets a conciliatory, but useless, $+\pi/8$ from General Excellence

Zl bcvavba: uvtuyl qhovbhf;
pregnvayl abg n ohfvarff gb trg vagb(*).

(*) Toggle ROT-13 in Vim with "ggVGg?" (as you might expect)

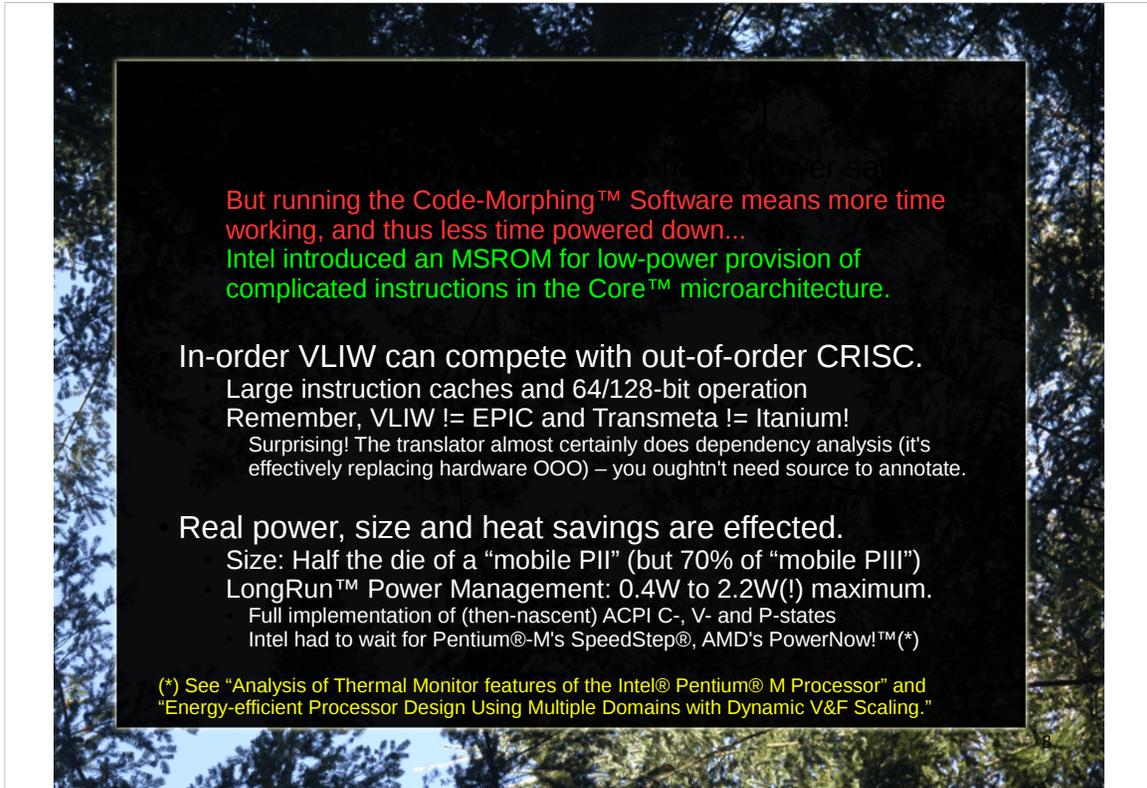


A straw poll!

Can a translator beat a compiler?

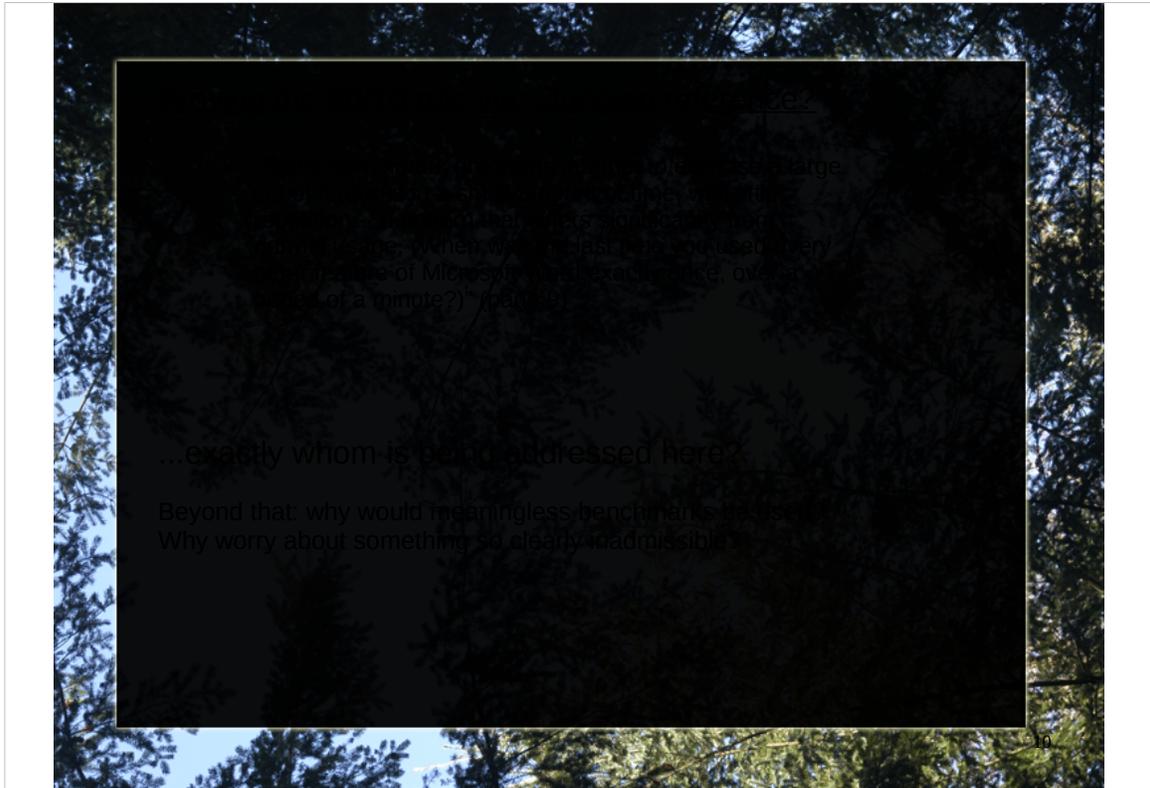
Show of hands?

My opinion: highly dubious;
certainly not a business to get into.



Note that the awesome graphic is comparing a **desktop** PIII to the Crusoe, whereas the (first) size table is all mobile comparisons. Tricky!

- “Closed-source” often means “legacy” :(



Note that the awesome graphic is comparing a **desktop** PIII to the Crusoe, whereas the (first) size table is all mobile comparisons. Tricky!

- “Closed-source” often means “legacy” :(



AT FIRST I WAS LIKE



BUT THEN I WAS LIKE



Let us return to Klaiber 1990, aka BULLS

...the compiler optimizer...
...invariant...
...settings...
...code...
...code...
...code...

...here, by virtue of the well-known, we can assume such
optimizations have been performed during the original compilation.

This exemplifies optimization that a hardware-only x86
implementation cannot do. A software-based translation
system can actually eliminate items from the instruction
stream, rather than just code them. (ibid)

Is Klaiber telling a baldfaced lie? Does he suffer from delusions
regarding compatibility? I think I'm pretty sure that he does.

(* Exhibits A and B: "Macro-" and "Micro-fusion". *Intel 64 and IA-32 Software Optimization Guide*, from the Core™ microarchitecture onwards.

...and, is there really no compilation context preserved across process lifetimes? Have fun, "find / -exec foo {} \;" (**)

This is why we have `mprotect(2)` and `mremap(2)`.

Instruction caches have the same issues.

...and, is there really no compilation context preserved across process lifetimes? Have fun, "find / -exec foo {} \;" (**)

(*) It was the NX bit which Intel added later; write protection has existed for five presidential administrations. (**) For this and other reasons, we use "find / -print0 | xargs -0 foo", right?