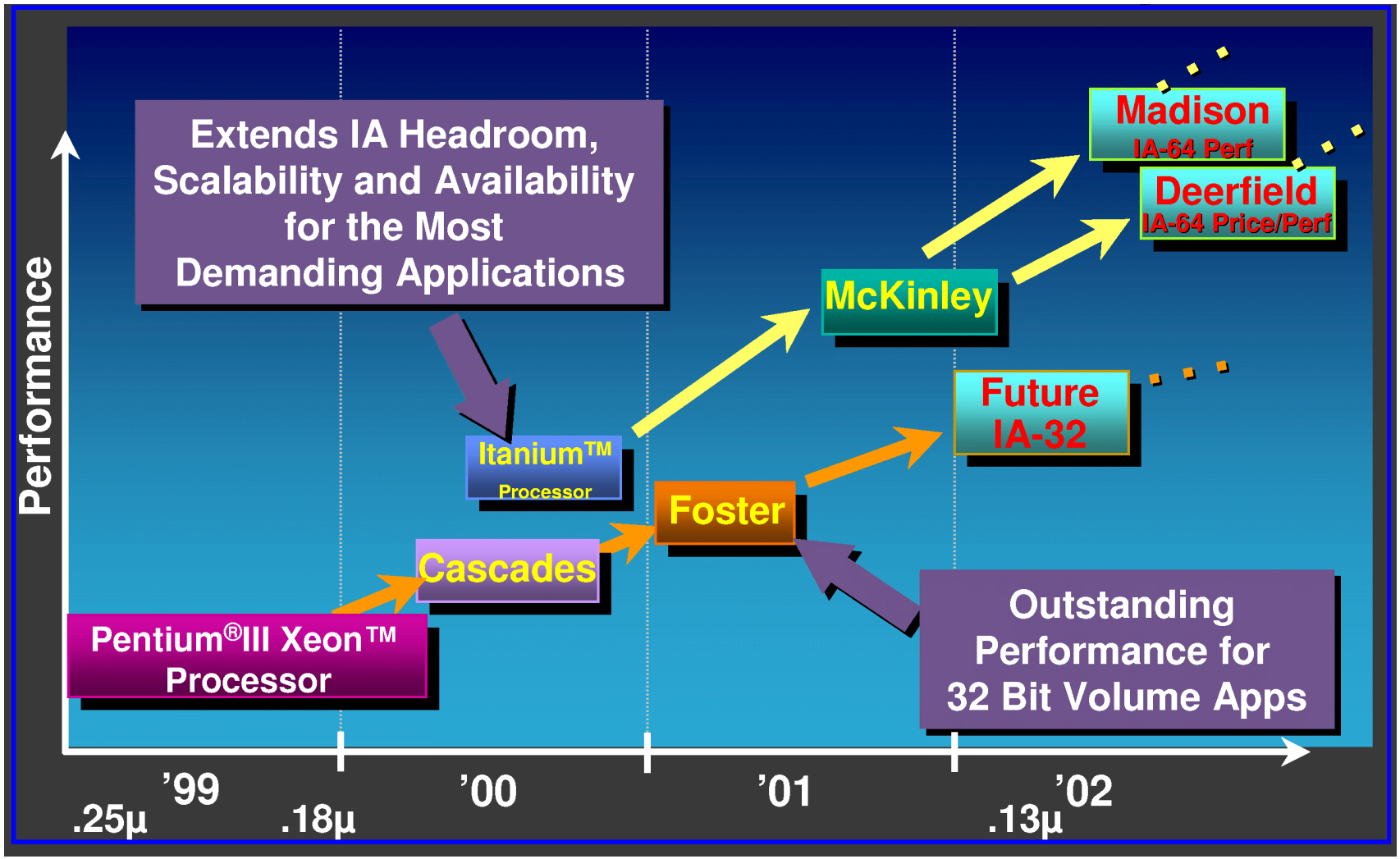
The EPIC architecture and a noptimization

Jacques-Olivier Haenni
EPFL - DI - LSL
October 25th, 2000

Presentation outline

- The IA-64 architecture
  - history and facts
- Superscalar, VLIW, and EPIC processors
- Itanium
  - some implementation details
- A noptimization
  - adding **nop**s to a program may improve its performance

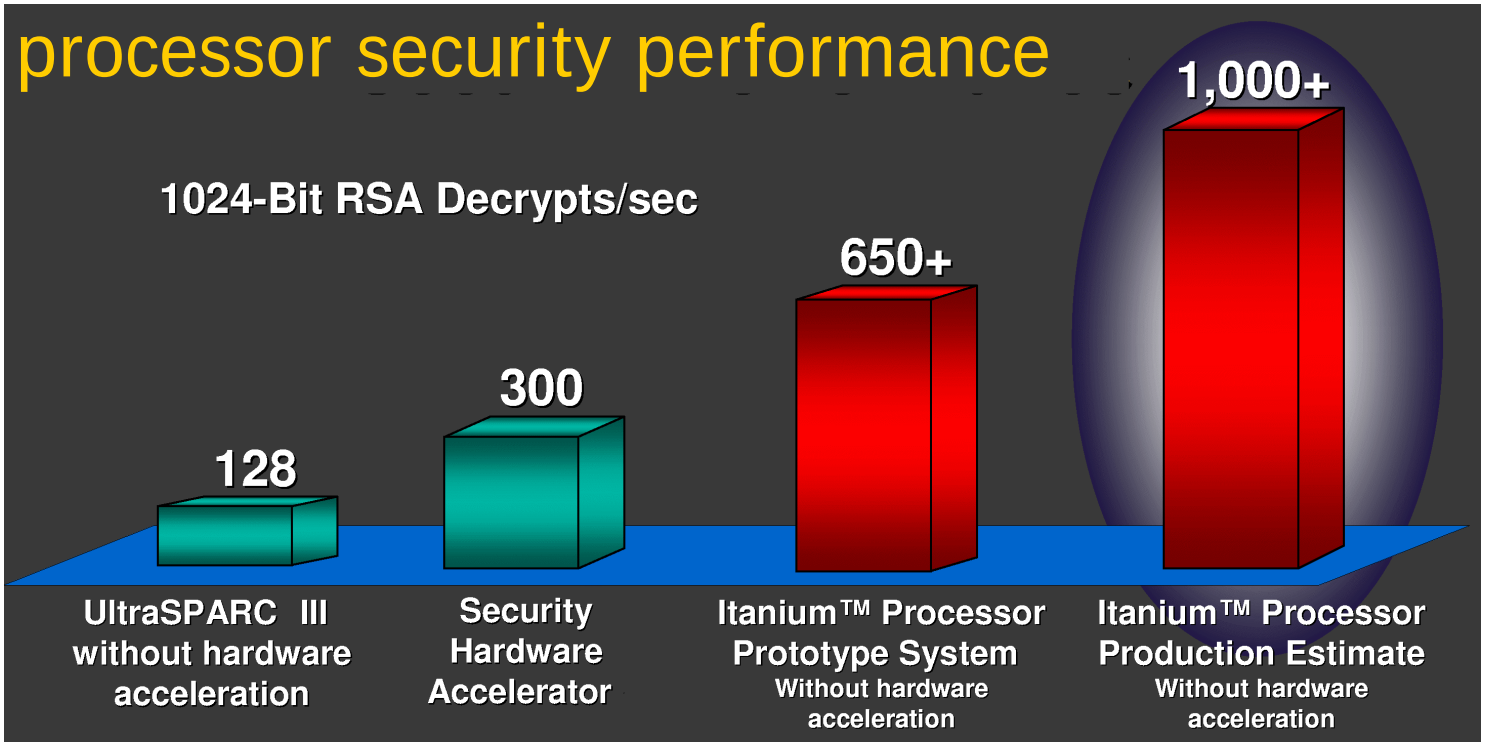
IA-64 roadmap



Source: "Itanium™ Processor Overview and the IA-64 Roadmap", Intel, March 2000

IA-64 performance

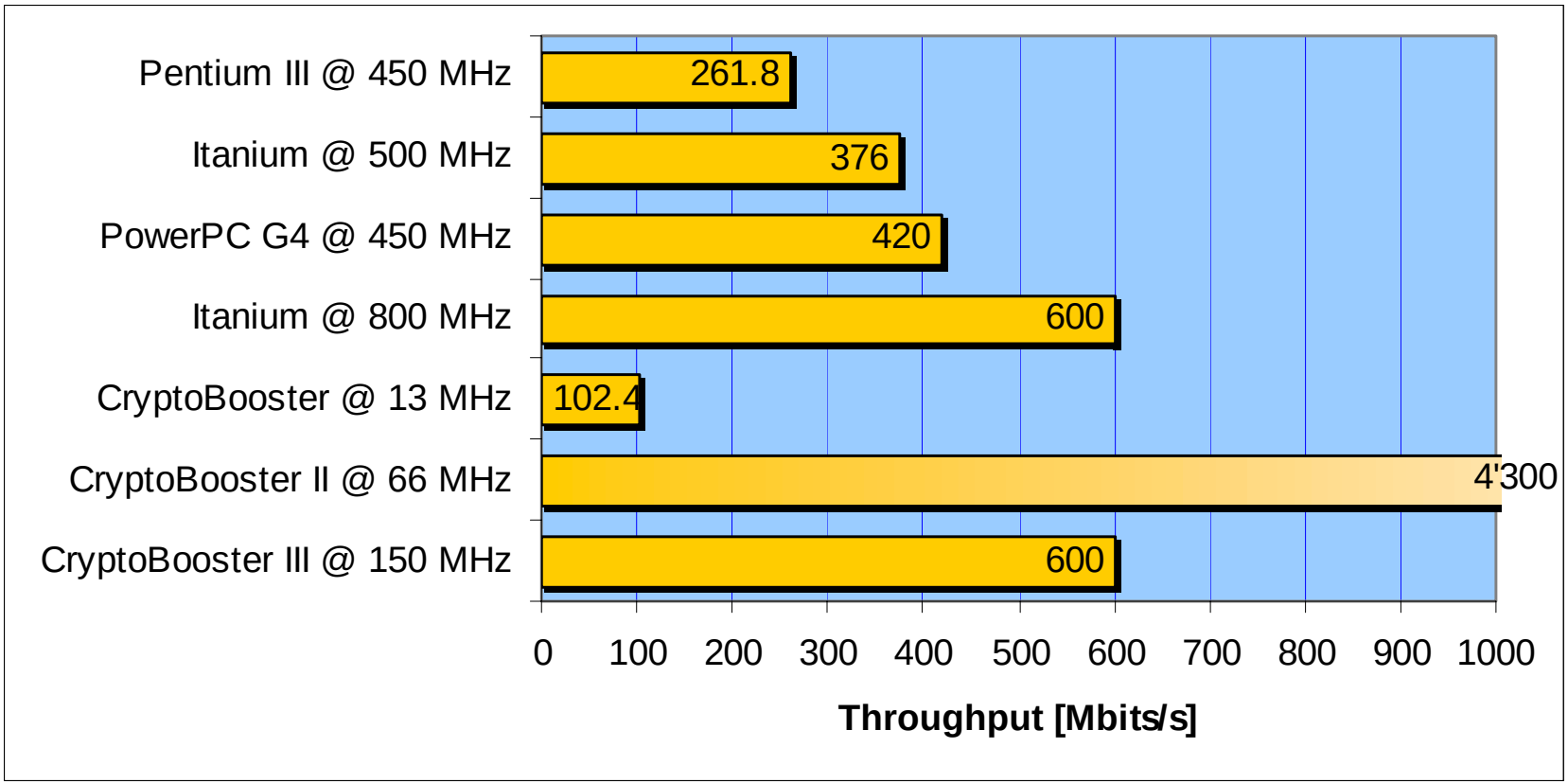
- Itanium @ 800 MHz vs. UltraSparc III @ 750 MHz:
  - 43% faster for transaction processing
  - 27% faster for floating point processing
- Itanium™ processor security performance



Source: "Itanium™ Processor Overview and the IA-64 Roadmap", Intel, March 2000

IA-64 performance

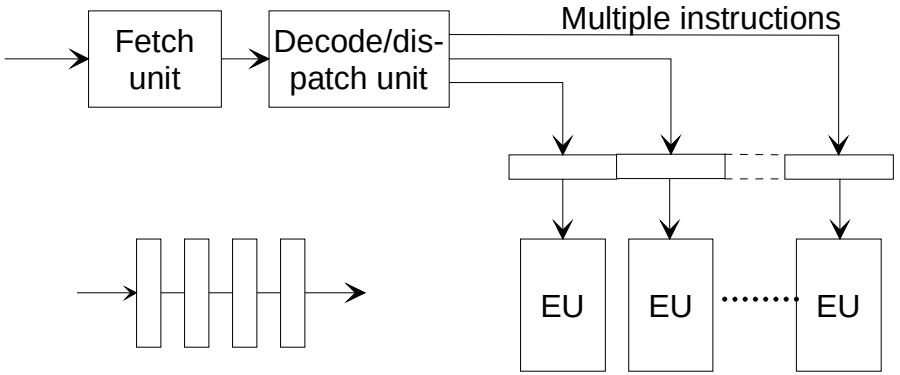
- IDEA encryption algorithm



Sources: Helger Lipmaa (Pentium III), LSL (Itanium, G4, CryptoBooster)

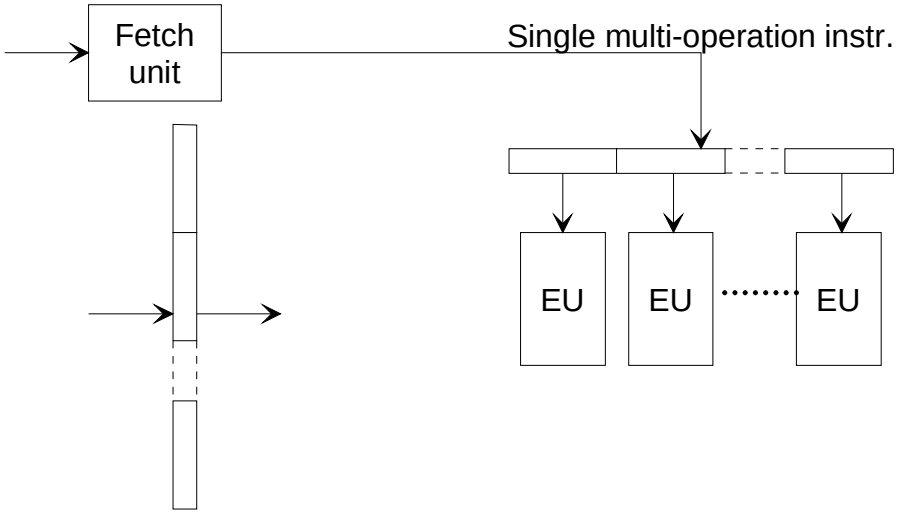
Superscalar vs. VLIW

Superscalar



- dynamic scheduling
- complex hardware
- can adapt to variable load lat.
- compact code

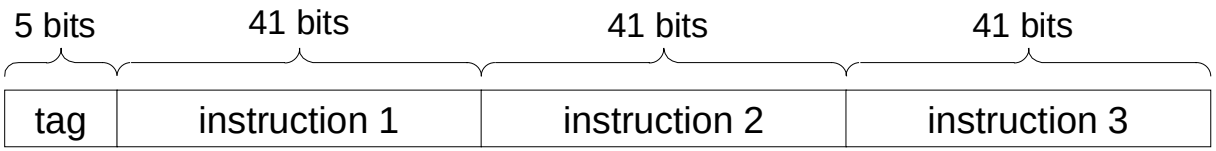
VLIW



- static scheduling
- simplier hardware
- uses a worst-case scheduling
- additional NOPs (waste of memory space and bandwidth)

EPIC and VLIW

- EPIC: Explicitly Parallel Instruction Computing
  - code "compression" by removing some additional NOPs
  - instruction bundle:



  - the tag, or template, describes:
    - the type (branch, load/store, integer, floating point,...) of the instructions
    - the dependences between the instructions
    - the dependence between this bundle and the next one
  - extended instructions

EPIC: code example

```
M/I  add   r1 = r2, r3
M/I  add   r4 = r5, r6      ;;    MI|I
M/I  sub   r7 = r8, r1

 M   ld4   r57 = [r56]
 F   fadd  f10 = f12, f13         MFI
 I   nop.i
 M   nop.m
 F   fadd  r14 = f15, f16         MFI
M/I  add   r16 = r18, r19

 M   st4   [r4] = r67        ;;
M/I  add   r24 = r56, r57         M|MI
M/I  add   r28 = r57, r58
```

EPIC advantages and drawbacks

- EPIC advantages (vs. VLIW)
  - reduced code size
  - the same code can be executed on different processor implementations (e.g. different number of functional units)
- EPIC drawbacks
  - cannot manage cache misses/hits in a flexible way
  - requires a more complex decode/dispatch unit than VLIW

Itanium and DEVIL

[DEVIL's Instruction Word]

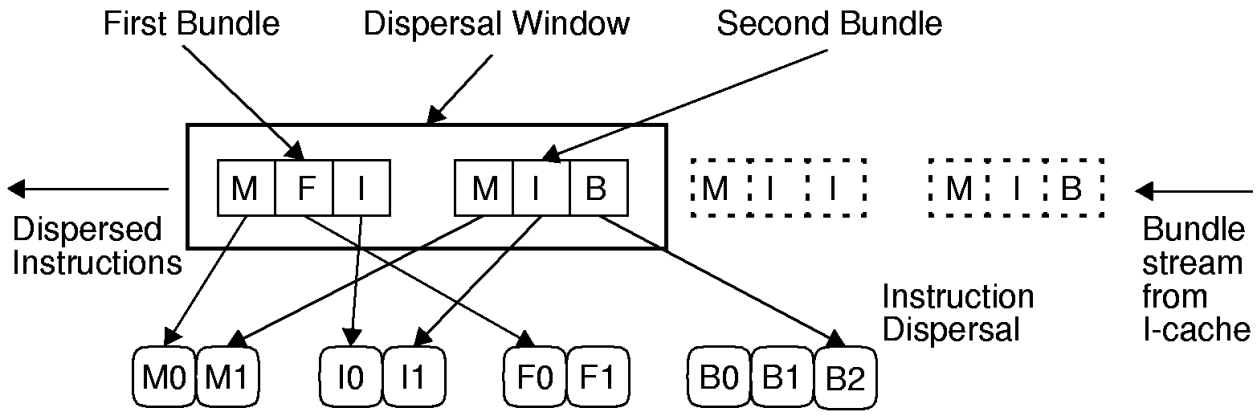
Fetched Instruction

tag	
0 0	One 30-bit instruction
0 1	Two 15-bit instructions in parallel
1 0	Two 15-bit instructions sequentially
1 1	Two 15-bit instructions seq., next in parallel

Source: "Instruction-level parallelism for Low-power Embedded Processors", Jean-Michel Puiatti, June 1999

Itanium

- Hardware architecture
  - functional units
    - 2 memory units
    - 2 integer units
    - 2 floating points units
    - 3 branch units

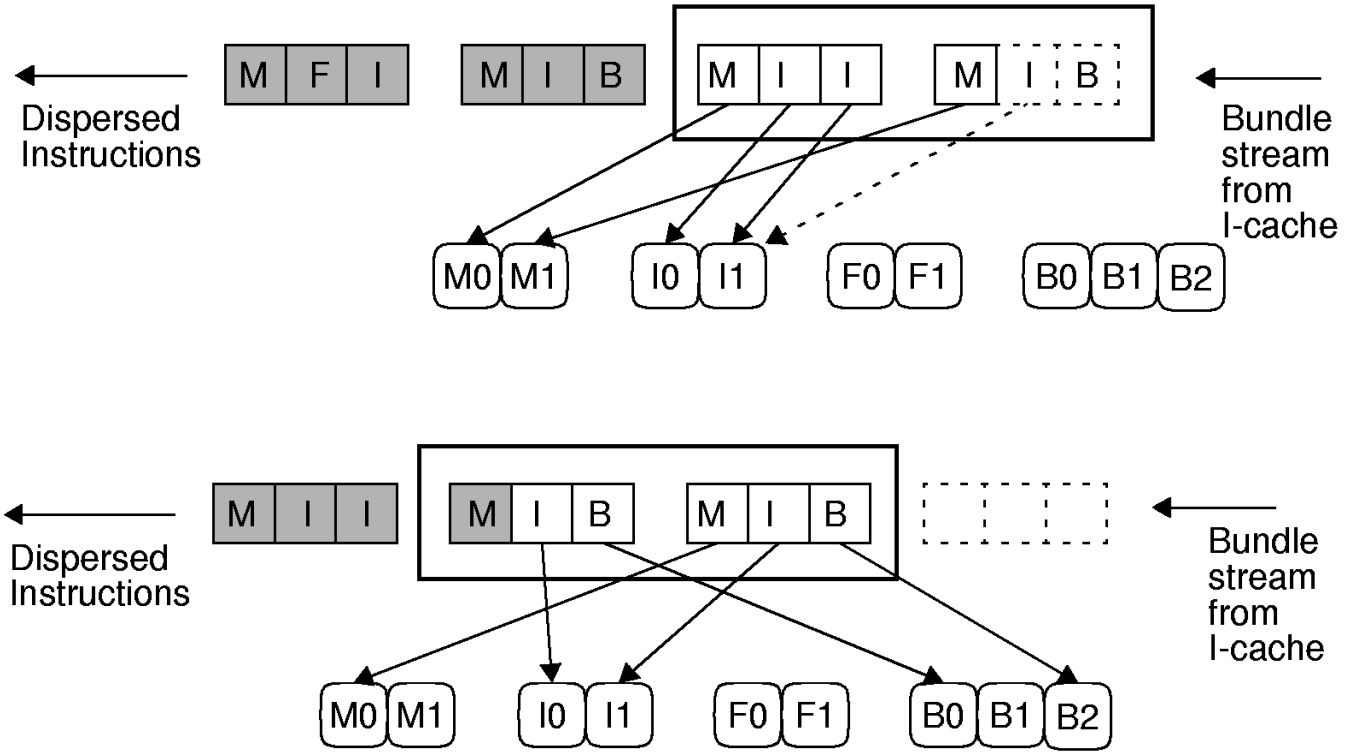


  - size of dispersal window
    - 2 bundles
  - very simple dispatching policy
  - up to 6 new instructions are executed per clock cycle
    (Intel says that "*EPIC enables up to 20 operations per clock*")

Source: "Itanium™ Processor Microarchitecture Reference", Intel, August 2000

Itanium

- Instruction dispatch example



Source: "Itanium™ Processor Microarchitecture Reference", Intel, August 2000

Itanium

- Instruction latencies
  - published by Intel, "verified" on a prototype machine
  - depend on source and target instructions

```
add r32 = 4, r32 ;;  // cycle 0
and r33 = r32, r33   // cycle 1
```

```
add r32 = 4, r32 ;;  // cycle 0
ld4 r33 = [r32]      // cycle 2
```

  - very long bypass latency between a multimedia and an integer instruction

```
and   r35 = r32, r33 ;; // cycle 0
and   r35 = r35, r34 ;; // cycle 1
padd2 r37 = r35, r36    // cycle 4
```

```
padd2 r35 = r32, r33 ;; // cycle 0
padd2 r35 = r35, r34 ;; // cycle 2
and   r37 = r35, r36    // cycle 14
```

Itanium: bypass latencies

Source Instruction Class	Target Instruction Class	Total Latency
IALU (for I slot instructions only)	LD,ST/address register	IALU + 1
ILOG	LD,ST/address register	ILOG + 1
LD	LD,ST/address register	LD + 1
IALU, ILOG	MMMUL, MMSHF, MMALU	IALU+2, ILOG+2
LD	MM operation	LD + 1
MM operations	IALU, ILOG, ISHF, ST, LD	If scheduled <4 cycles apart, 10 clocks; If schedule >=4 cycles apart) 4 clocks
TOBR, TOPR, TOAR (pfs only)	BR	0
FRBR, FRCR, FRIP, or FRAR (FRxx)	MMMUL, MMSHF, MMALU	FRxx + 1
FMAC	FMISC, ST, FRFR	FMAC + 2
SFxxx (32-bit parallel floating point)	Fxxx (64/82-bit floating point)	SFxxx + 2 cycles
Fxxx (64/82-bit floating point)	SFxxx (32-bit parallel floating point)	Fxxx + 2 cycles

Source: "Itanium™ Processor Microarchitecture Reference", Intel, March 2000

~~Joke~~...

- Here are two loops; which one is the fastest ?

```
Loop1:
        padd2 r34 = r32, r33  ;;  //  1
        xor   r36 = r34, r35      // 13
        br    Loop1
```

```
Loop2:
        padd2 r34 = r32, r33  ;;  //  1
        nop                   ;;  //  2
        nop                   ;;  //  3
        nop                   ;;  //  4
        xor   r36 = r34, r35      //  5
        br    Loop2
```

- measured duration:
  - Loop1 : 16 cycles
  - Loop2 : 5 cycles

A noptimization

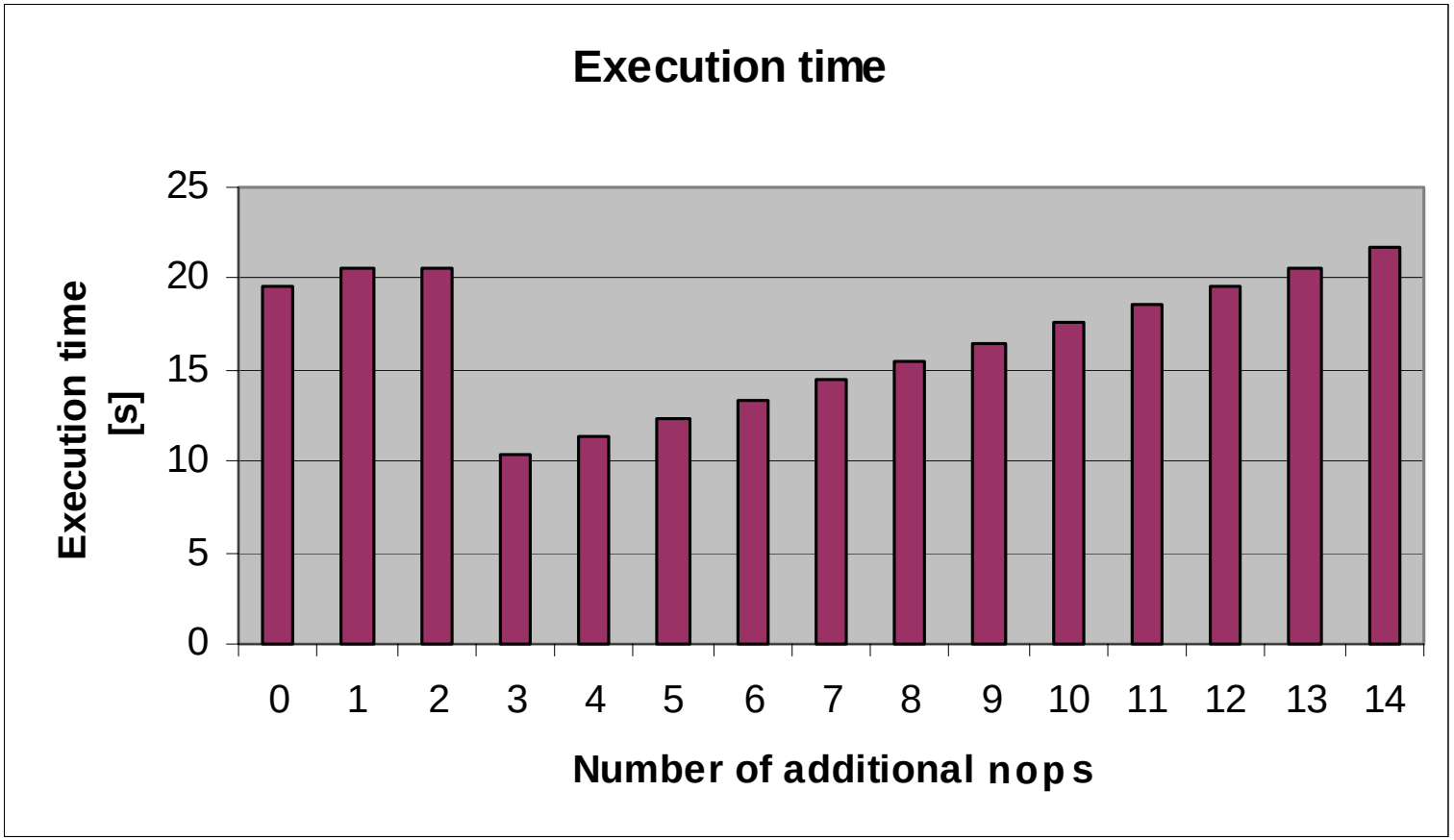
- Example
  - Compiler: gcc -O4

```
long toto (int shift,
           long long* tbl,
           int num)
{
  int i;

  for (i = 0; i < num; i++)
    tbl[i] = (tbl[i] << shift) + 1;

  return 0;
}
```

```
.L6:
        shladd r16 = r15, 3, r19
        ;;
        ld8 r14 = [r16]
        adds r15 = 1, r15
        ;;
        shl r14 = r14, r18
        sxt4 r15 = r15
        ;;
        adds r14 = 1, r14
        cmp4.gt p6, p7 = r17, r15
        ;;
        st8 [r16] = r14
  (p6) br.cond.dptk .L6
```

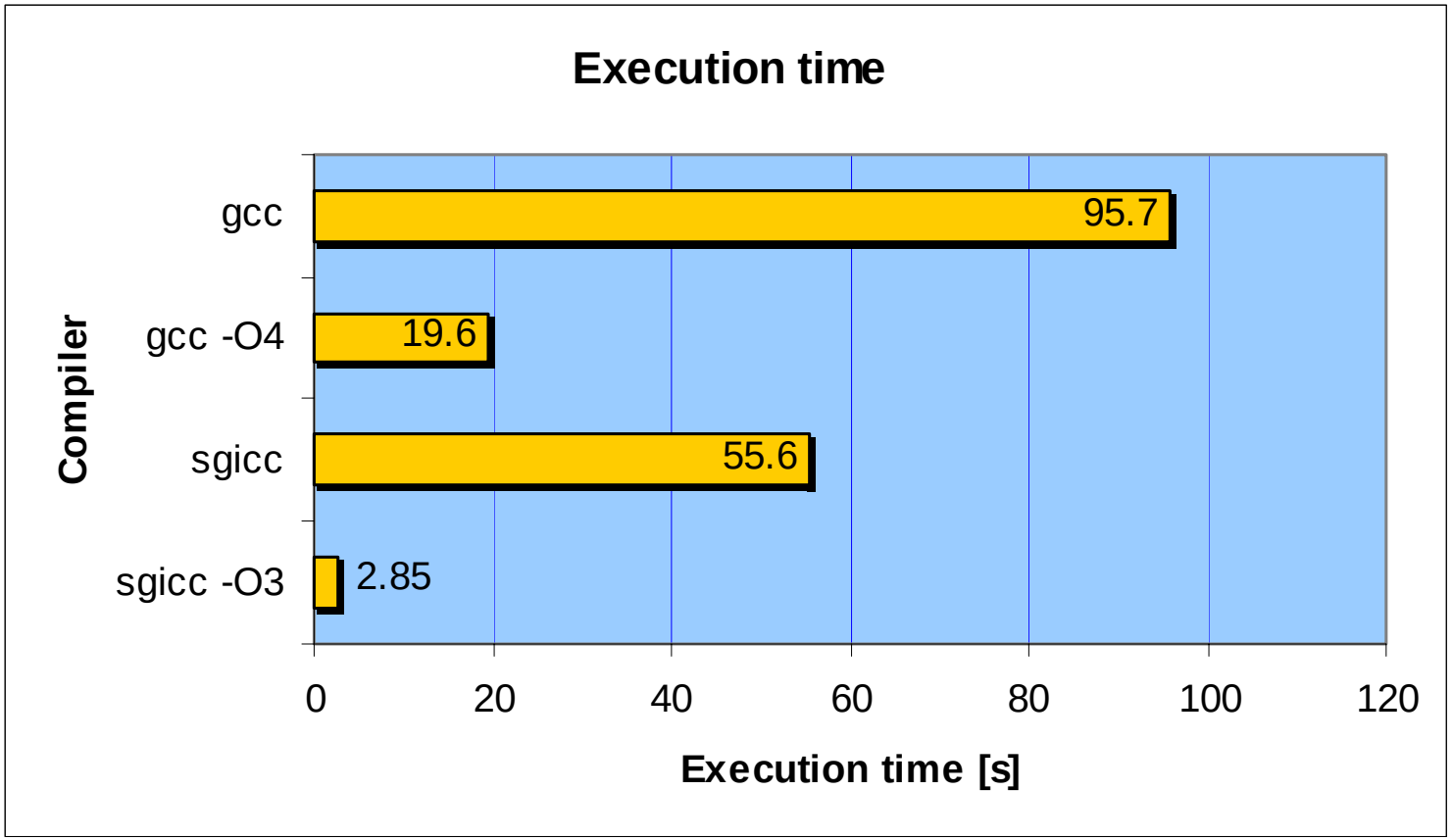
A noptimization

- Results



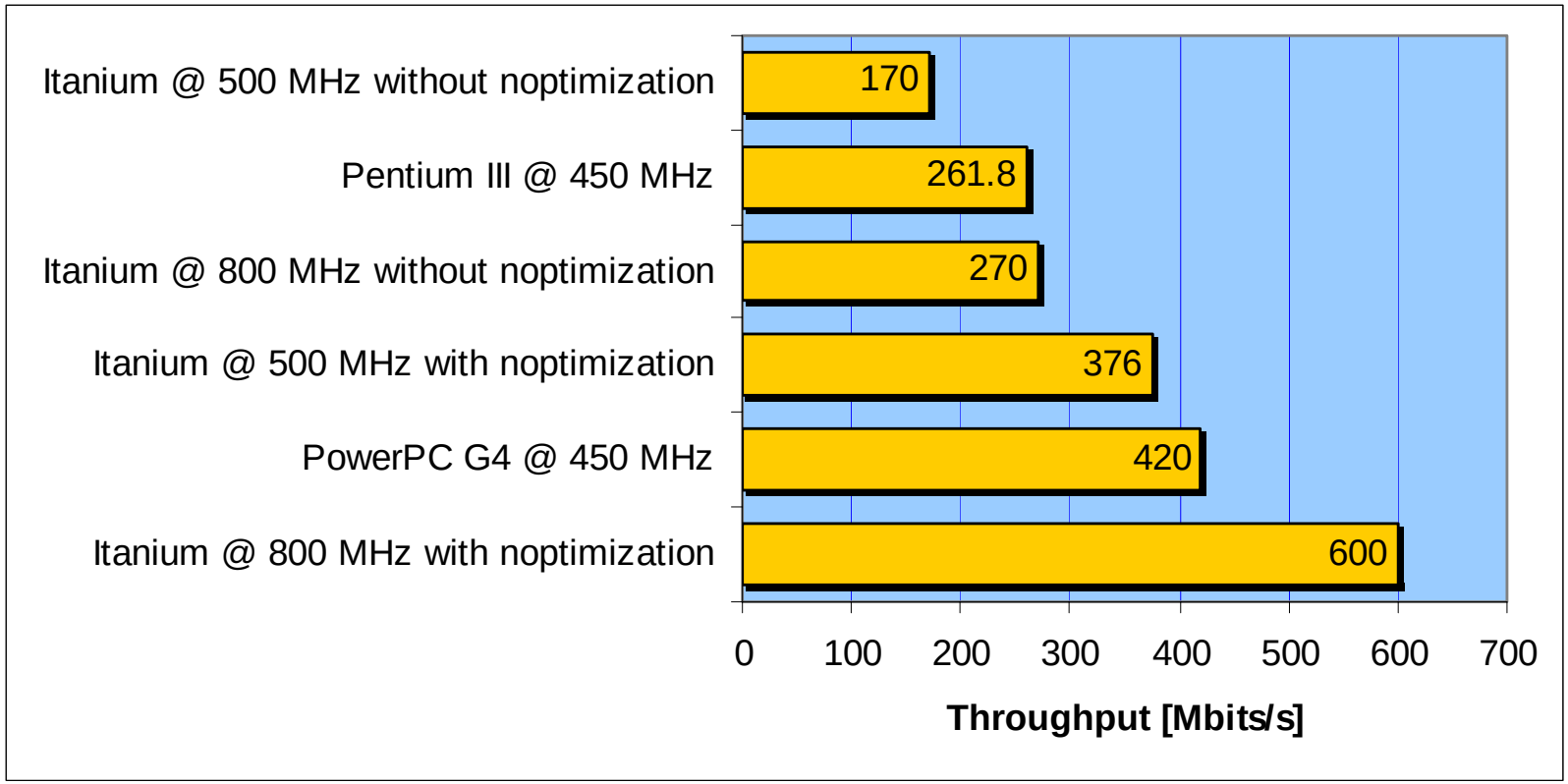
A noptimization

- Compiler comparison: gcc vs. SGI Pro64™



IDEA

- Effect of the noptimization



Conclusion

- Itanium is only the first implementation of IA-64
- IA-64 processors can execute IA-32 code
- Itanium and the LSL:
  - Itanium
    - predicated execution
    - instruction coding (EPIC)
  - LSL
    - picopascaline
    - DEVIL