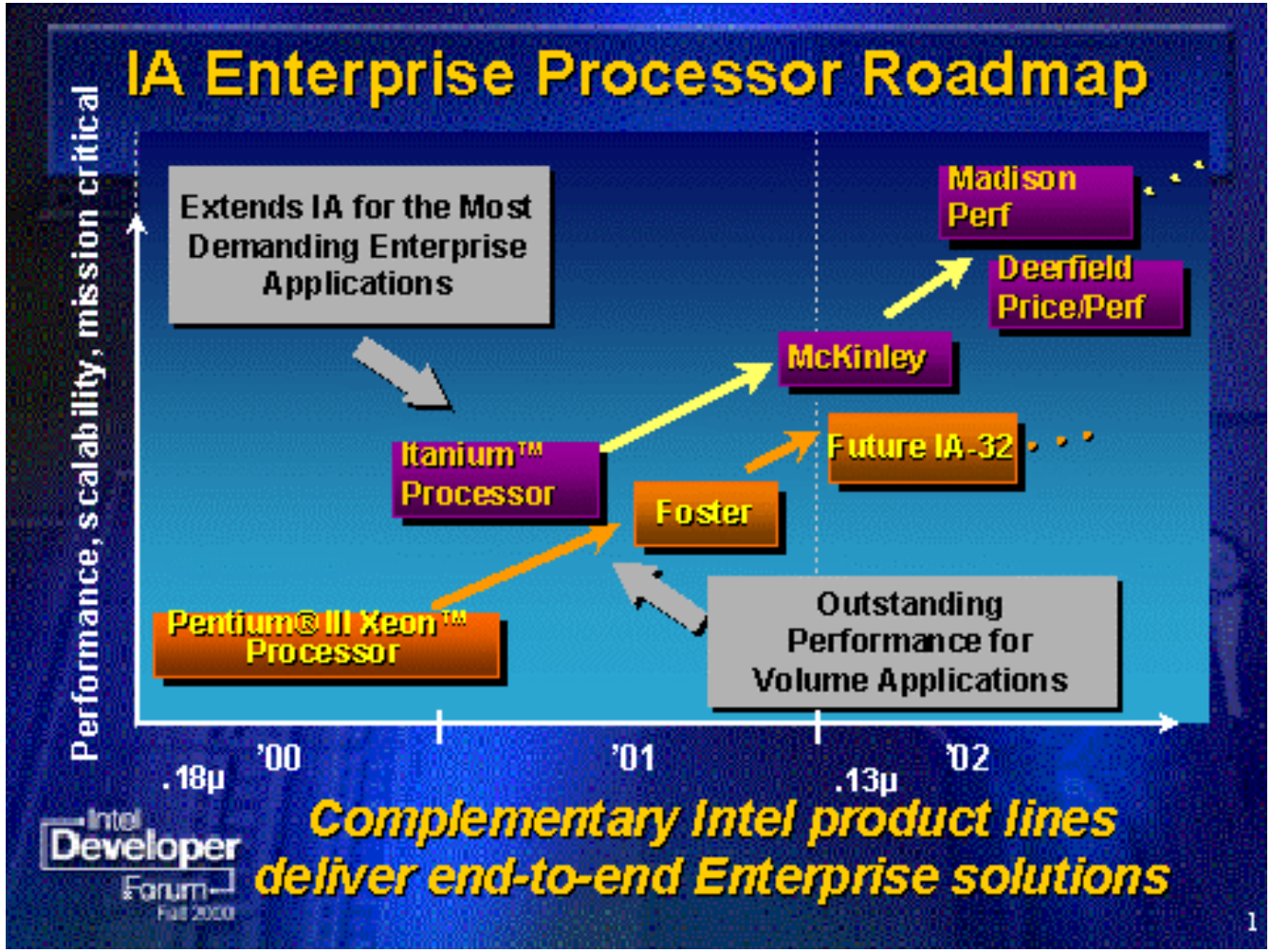
The benefits of EPIC architectures for multimedia applications

Jacques-Olivier Haenni
EPFL - DI - LSL
March 2nd, 2001

Presentation outline

- Introduction to Itanium and to cryptography
- IDEA: from simulation to practice
- A noptimization
  - description and potential benefits
  - compiler selection: Trimaran vs. SGI Pro64
  - implementation
  - applications
- Publications
- Conclusion and future work

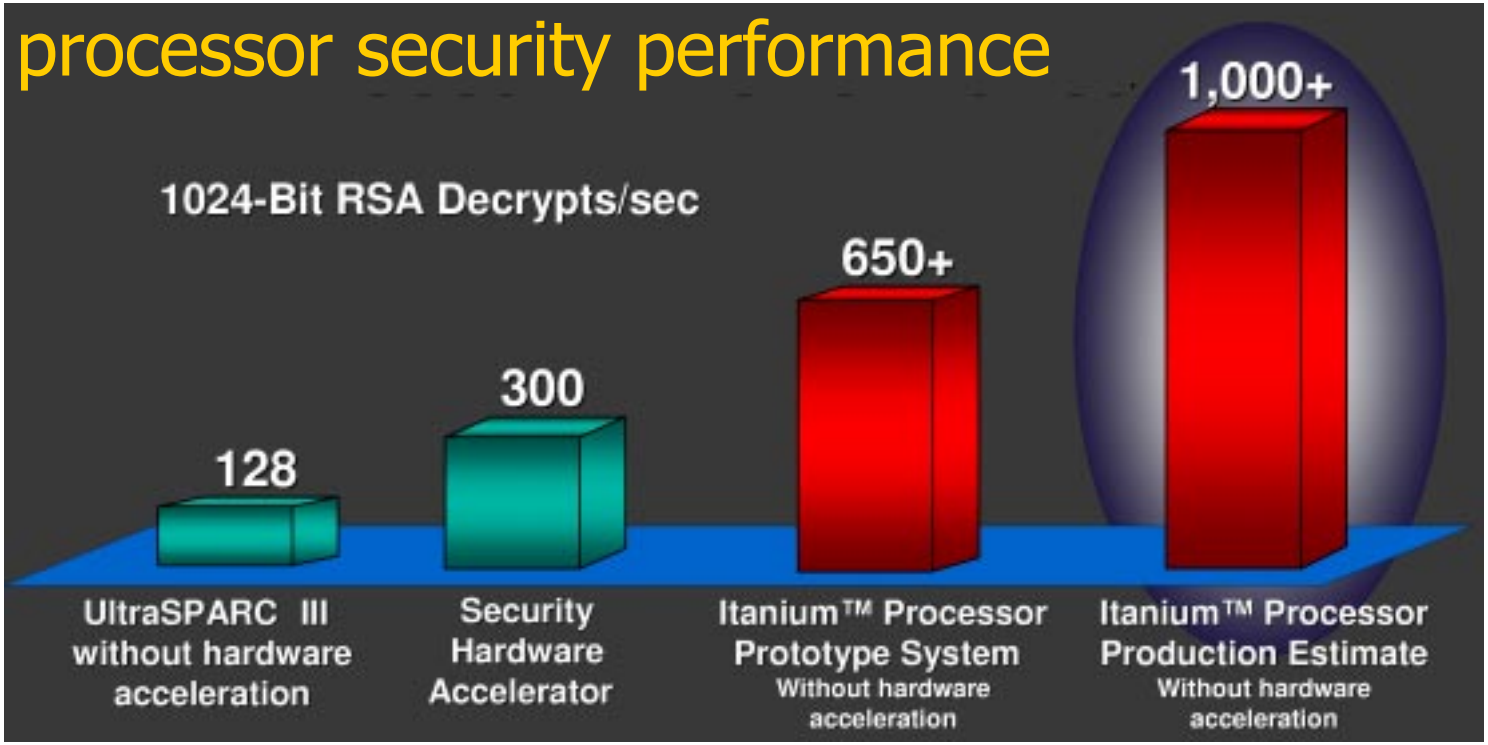
Itanium processor family (IPF)



Source: Intel web site

Itanium performance

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



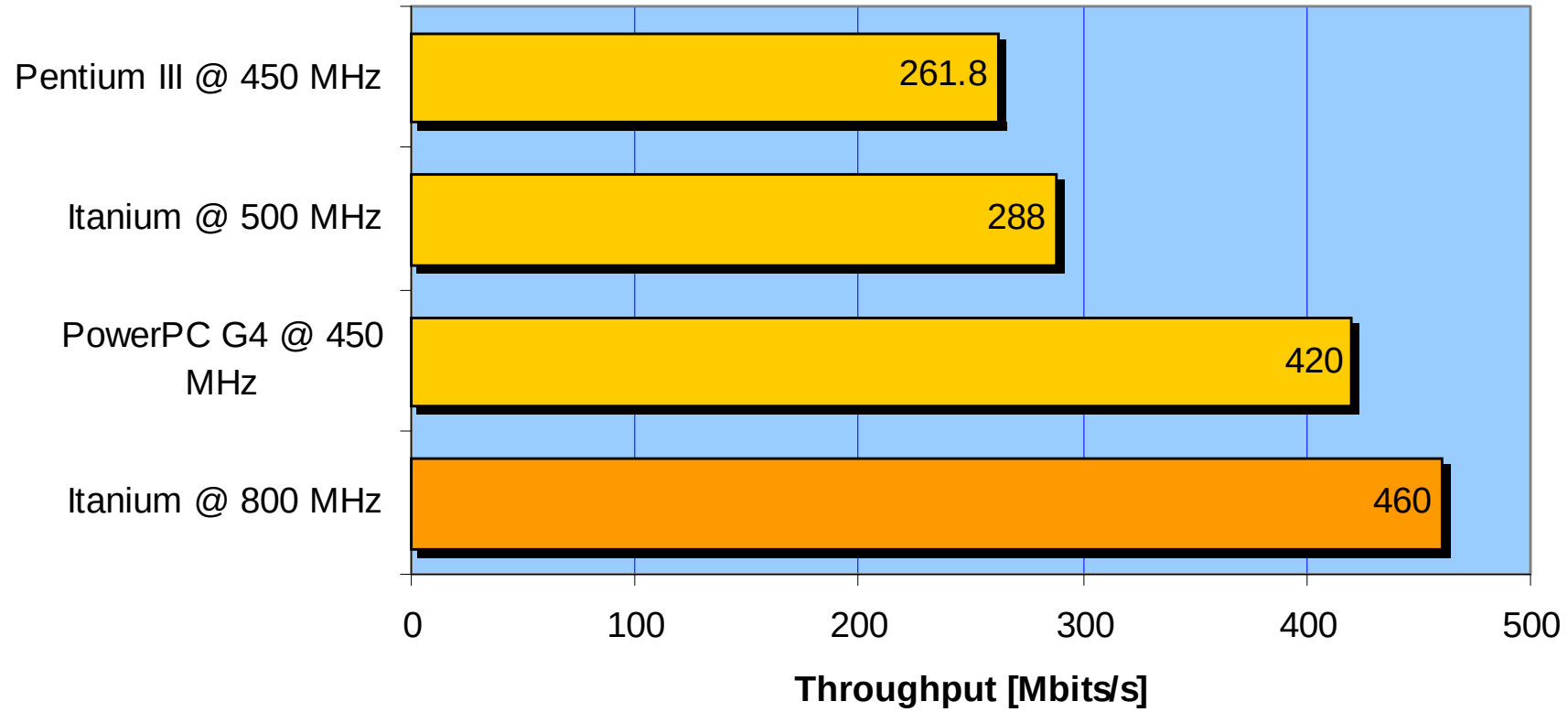
Source: "Itanium™ Processor Overview and the IA-64 Roadmap", Intel, March 2000

IDEA implementation

- Assumptions
  - execution of up to 2 bundles (6 instr.) per clock cycle
  - one clock latency for arithmetic (and multimedia) instructions except for the multiplication (2 cycles)
- Functional simulation
- Performance estimation : 1’500 Mbits/s @ 800 MHz
  - Pentium III : 260 Mbits/s @ 450 MHz

IDEA implementation

- IDEA encryption throughput



Itanium

- V6 vs. Itanium
  - memory latencies ? no
  - ILP ? yes
    - only 4 arithmetic instructions per clock cycle
    - only one multimedia multiplication per clock cycle
  - instruction latencies ? yes
    - multimedia instructions: 2-cycle latency
  - and the bypass latencies between integer and multimedia instructions !!!

Itanium

- Instruction latencies
  - 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

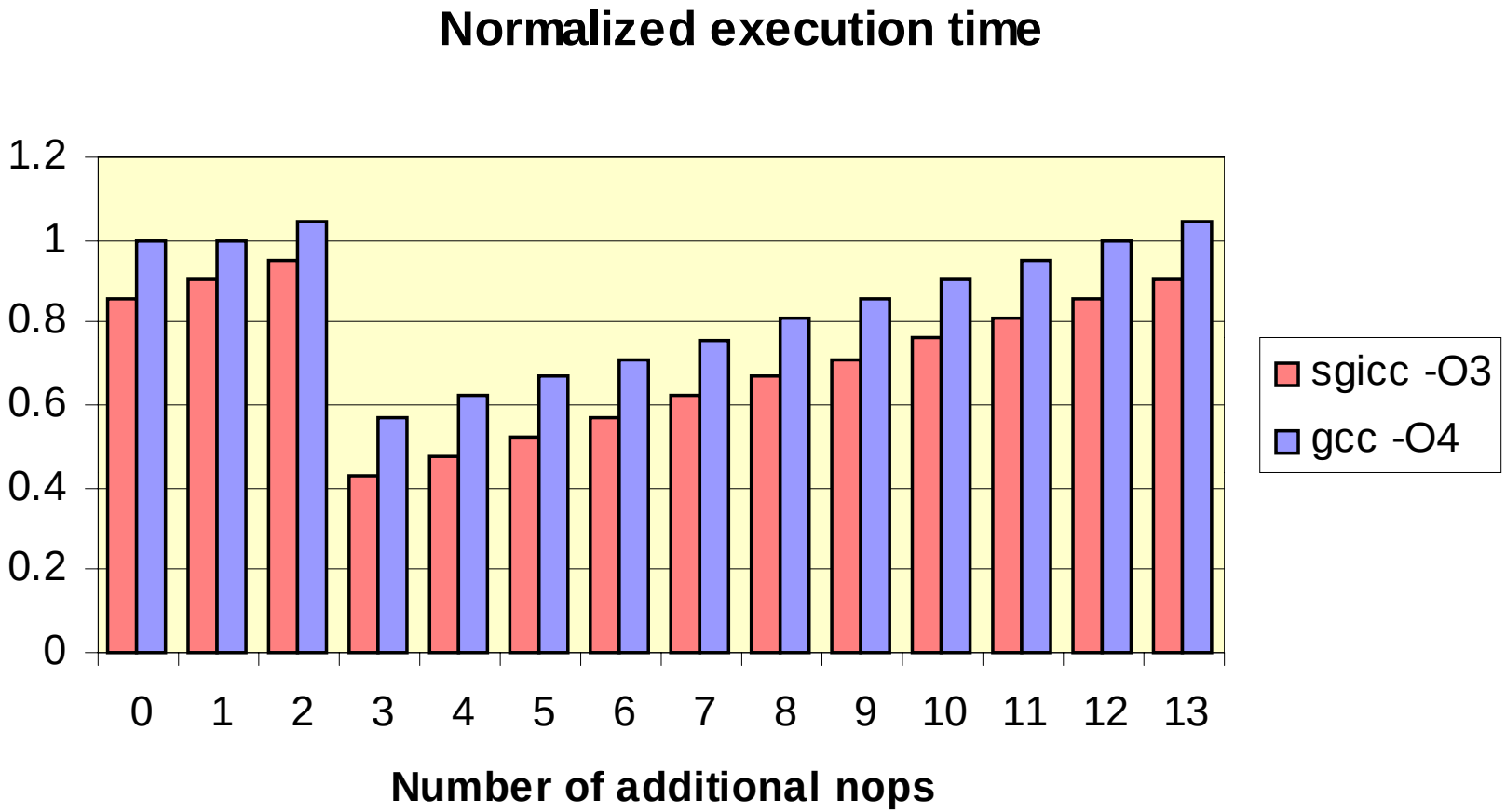
- Example
  - Compiler: gcc -O4

```
void rotate (int shift,
             unsigned long long* tbl,
             int num)
{
  for (int i = 0; i < num; i++)
    tbl[i] = (tbl[i-1] << shift)
             | (tbl[i-1] >> (64 - shift));
}
```

```
.L6:
        [...]
        ;;
        ld8 r14 = [r16]
        cmp4.gt p6, p7 = r19, r18
        ;;
        shr.u r15 = r14, r20
        shl r14 = r14, r21
        ;;
        or r14 = r15, r14
        ;;
        st8 [r17] = r14
  (p6) br.cond.dptk .L6
```

A noptimization

- `rotate` function

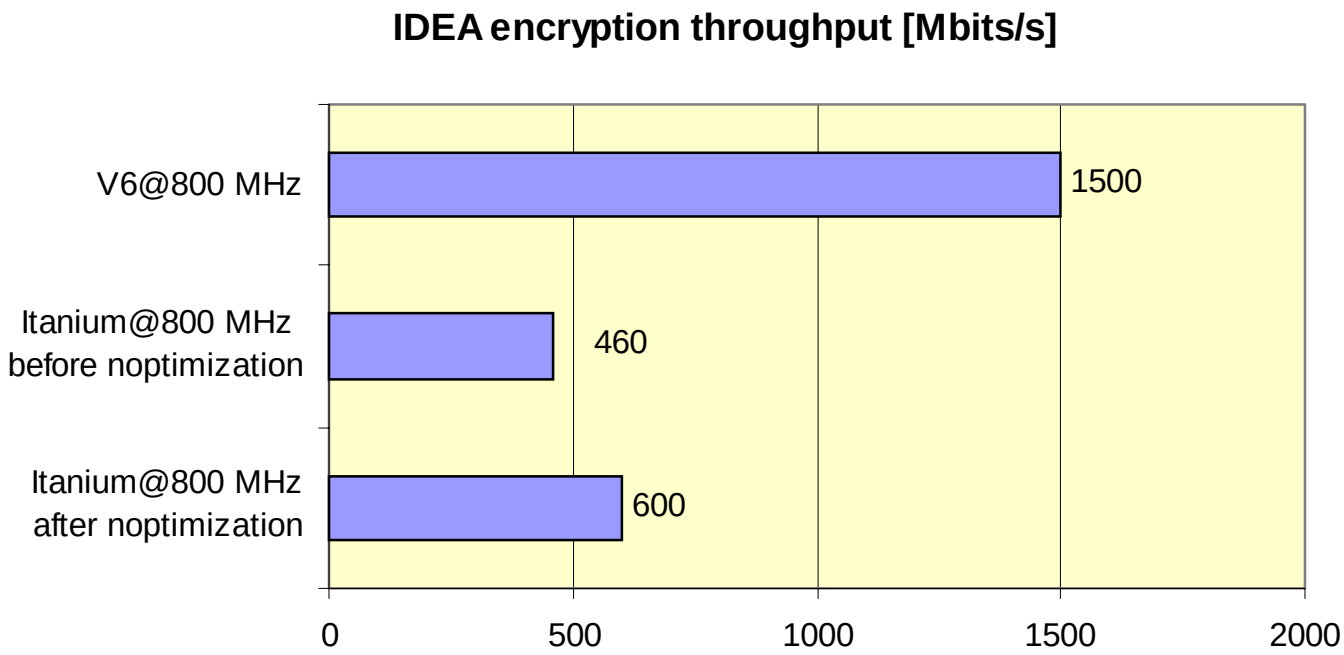


A noptimization

- When can it be profitable ?
  - variable shifts/rotates
  - 32-bit integer multiplication
    - gcc:
      - converts the values into floating-point,
      - performs the multiplication,
      - and converts back the result into integer
    - SGI Pro64 (sgicc):
      - uses the `pmpyshr` instruction which computes 16 bits of the product of two 16-bit values:
        $(x_1 \cdot 2^{16} + x_0) \cdot (y_1 \cdot 2^{16} + y_0)$
        $= x_1 \cdot y_1 \cdot 2^{32} + x_1 \cdot y_0 \cdot 2^{16} + x_0 \cdot y_1 \cdot 2^{16} + x_0 \cdot y_0$

A noptimization

- IDEA encryption algorithm
  - application of the noptimization:
    - nop addition
    - no code rescheduling

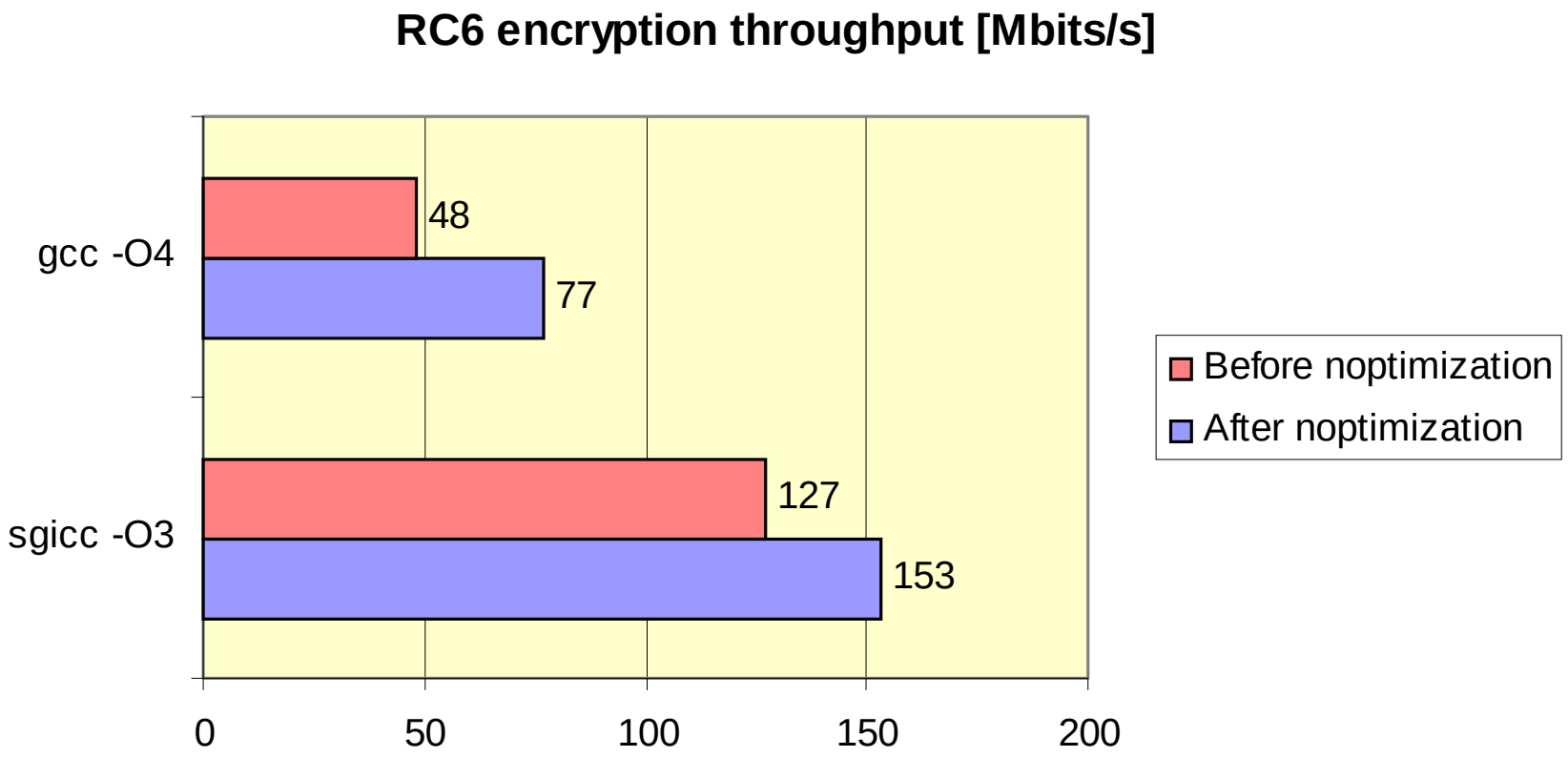


A noptimization

- RC6 encryption algorithm
  - main operations :
    - $X \cdot (2 \cdot X + 1) \bmod 2^{32}$
    - $X <<<_{32} 5$ (rotation by a fix number of bits)
    - $X <<<_{32} a$ (rotation by a variable number of bits)
    - $X \oplus Y$ (exclusive or)
    - $(X + Y) \bmod 2^{32}$
  - C implementation
  - compilers : gcc and SGI Pro64 (sgicc)
  - the noptimization is applied manually

A noptimization

- RC6 encryption algorithm



Compiler choice: SGI Pro64

- Trimaran (Impact)
  - very complex (simulator,…)
  - retargetable
  - no Itanium backend
  - a lot of documentation
  - discussion forum
  - many changes in the last two years
- SGI Pro64
  - only a compiler
  - Itanium specific
  - executable programs
  - not a lot of doc. (yet ?)
  - active mailing list
  - readable code, well-organized sources

Noptimization implementation

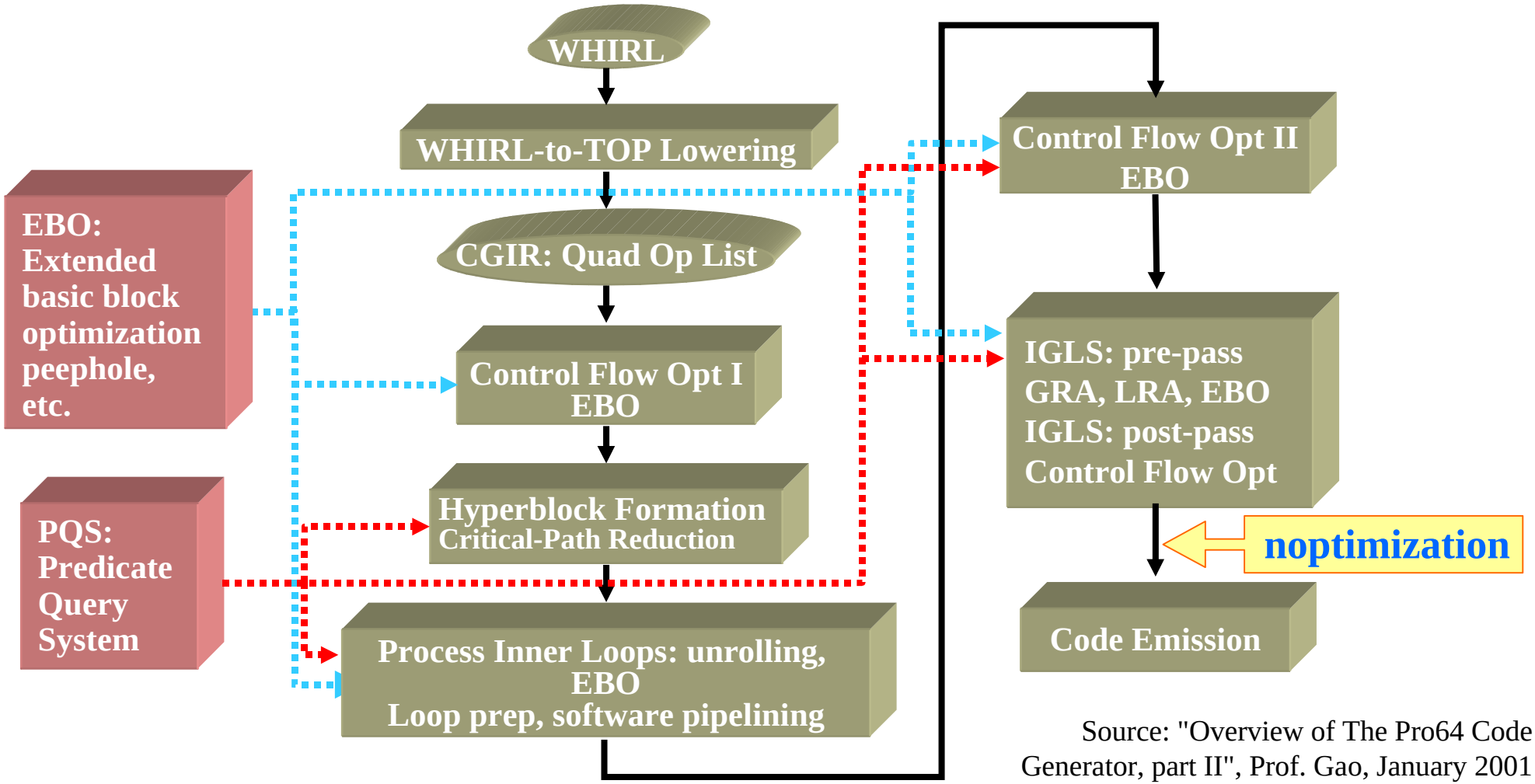
- Adding nops is not very clever…
- Modifying the scheduler seems better.

- But:
  - the scheduler already knows about the instruction latencies, but never adds “empty” cycles

- Therefore:
  - we’re going to add nops

Problems

- Flow chart of the Code Generator



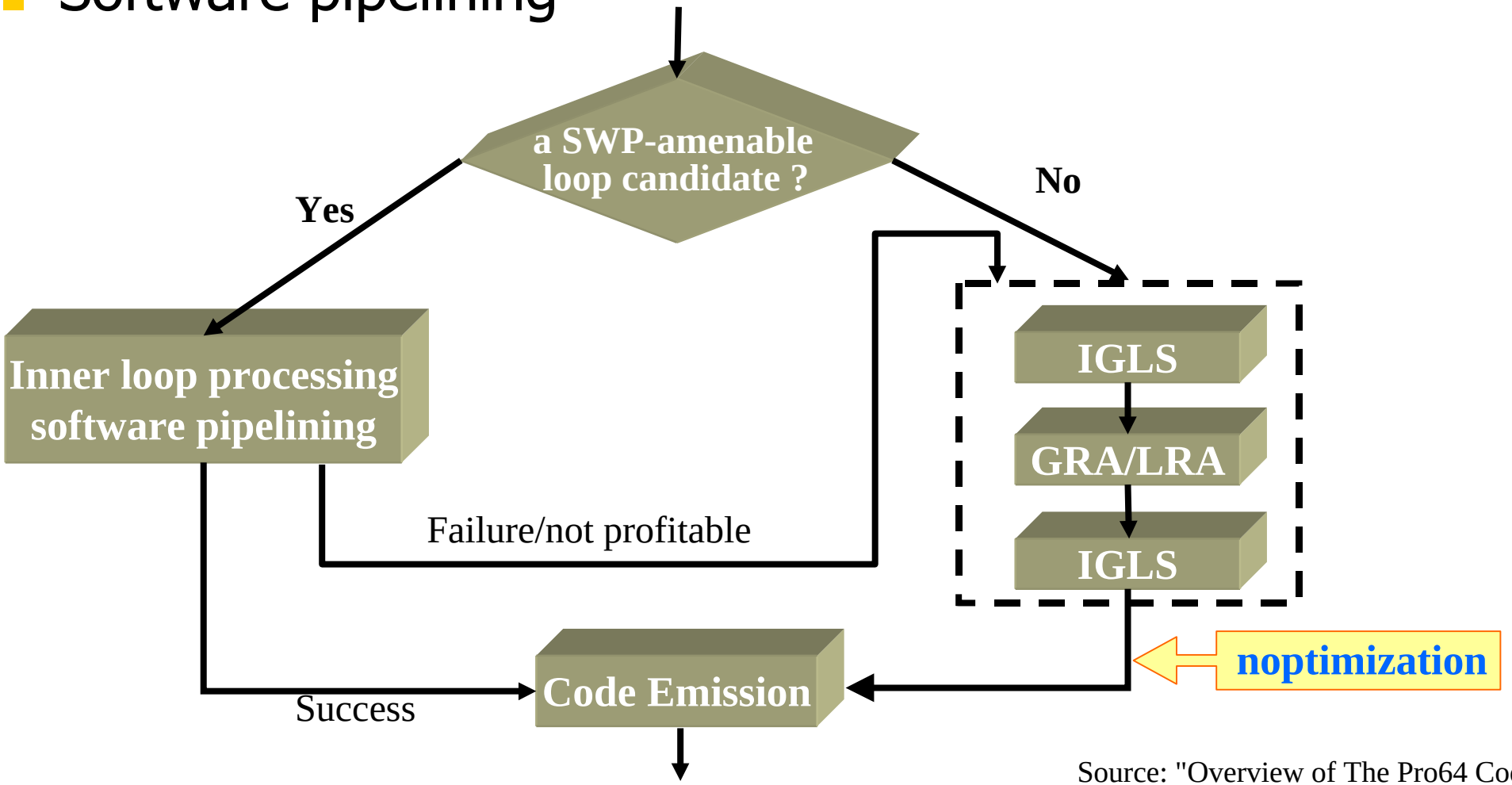
Source: "Overview of The Pro64 Code Generator, part II", Prof. Gao, January 2001

Problems

- Compilation problems
  - some modules are said to be compilable, but are not…
- Compiler problems
  - gcc and g++ have bugs
  - (so have sgicc…)
- Executable code generation
  - SGI Pro64 is compiled as a cross-compiler for IPF running on IA-32 machines
  - Linux PC at LSL: C to asm compilation
  - Itanium machine at Suse.de: assembler and linker

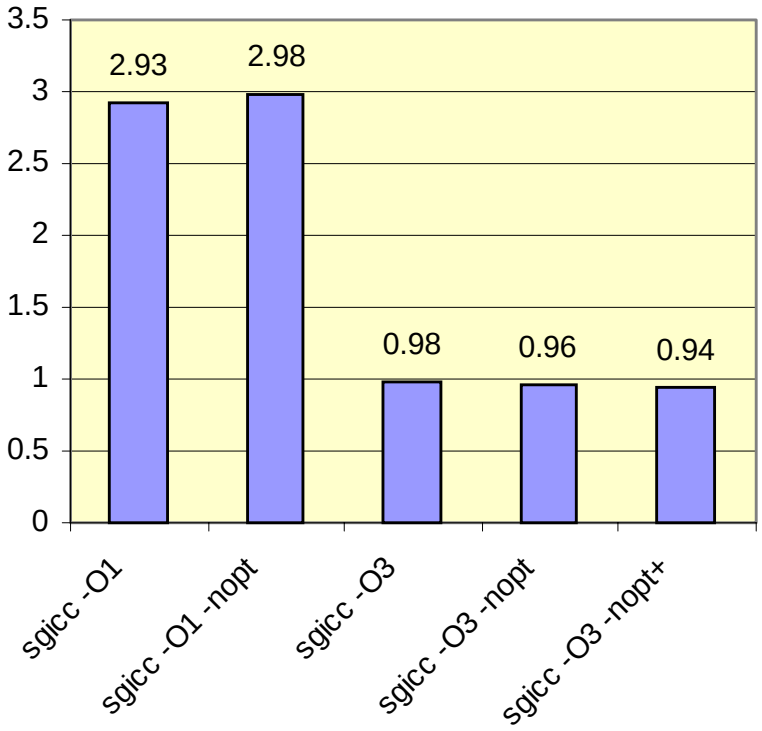
Problems

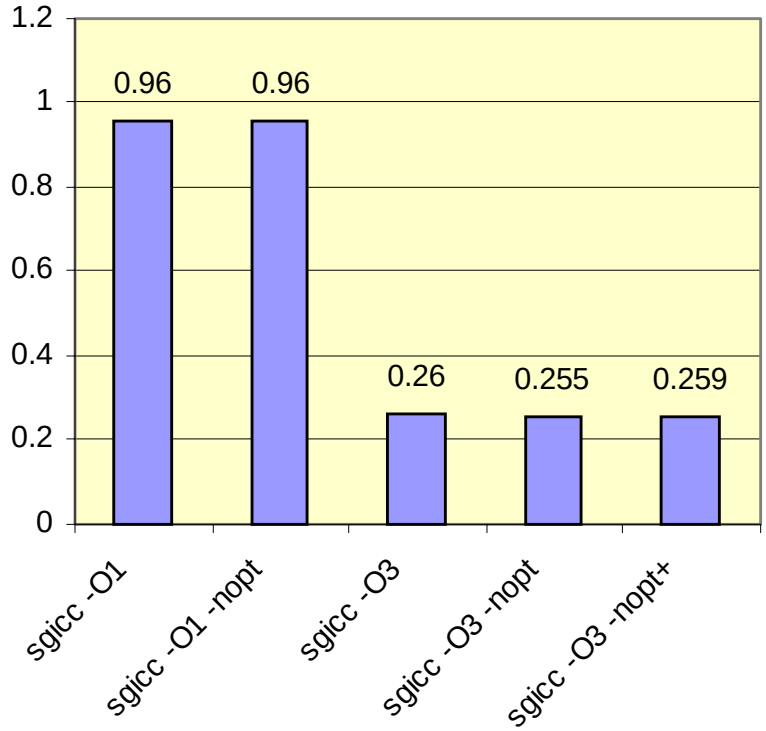
- Software pipelining



Source: "Overview of The Pro64 Code Generator, part II", Prof. Gao, January 2001

Noptimization: JPEG

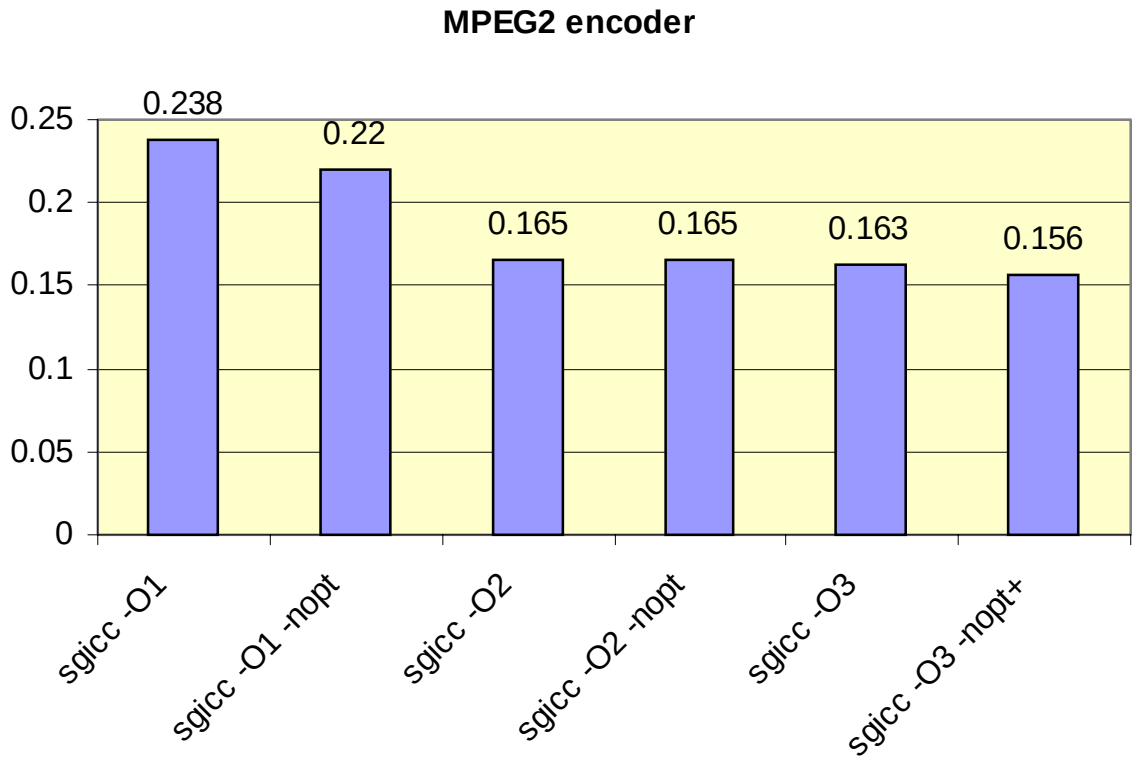




- main problem: lack of multimedia instructions

Noptimization

- MPEG2 encoder



  - and other benchmarks ???

Current publications

- SympA'7
  - 7ème Symposium sur les Architectures Nouvelles de Machines
  - Paris, April 24 – 27, 2001
  - "La noptimisation : une amélioration de performance pour *Itanium*"
- PACT'2001
  - International Conference on Parallel Architecture and Compilation Techniques
  - Barcelona, September 8 – 12, 2001

Future work

- Paper for PACT'2001
  - deadline: March 5th, with an automatic 1-week extension
- Support for multimedia instructions in SGI Pro64
  - main task until July or August
- PhD thesis editing
- Implementation of the noptimization in the SWP
  - this is not a priority…