1

MIPS Pro Compilers

2

- MIPS ABi
- MIPS ISA
- Target Processor
- Compiladores: ucode, ragnarok, mongoose
- Execution – compilation table
- MIPS Pro compiler overview
- WHIRL
- WHIRL levels
- Compiler flags & groups
- Compiler Optimisation Levels
- Distribución de datos
- Variable Page sizes
- Page replication – migration

MIPS App Binary Interface (ABI)

3

–> 32: 32 bit address

modelo viejo de 32 bits

–> n32: 32 bit address, 64 bit operations

modelo mixto de 32/64 bits, no porting (<4 GB direc.)

–> 64: 64 bit address, 64 bit operations

modelo 64 bits (> 4GB), requiere porting a 64 bits

MIPS ABI funciona en todos los sistemas MIPS ABI compliant (Pyramid, Sony, ...)

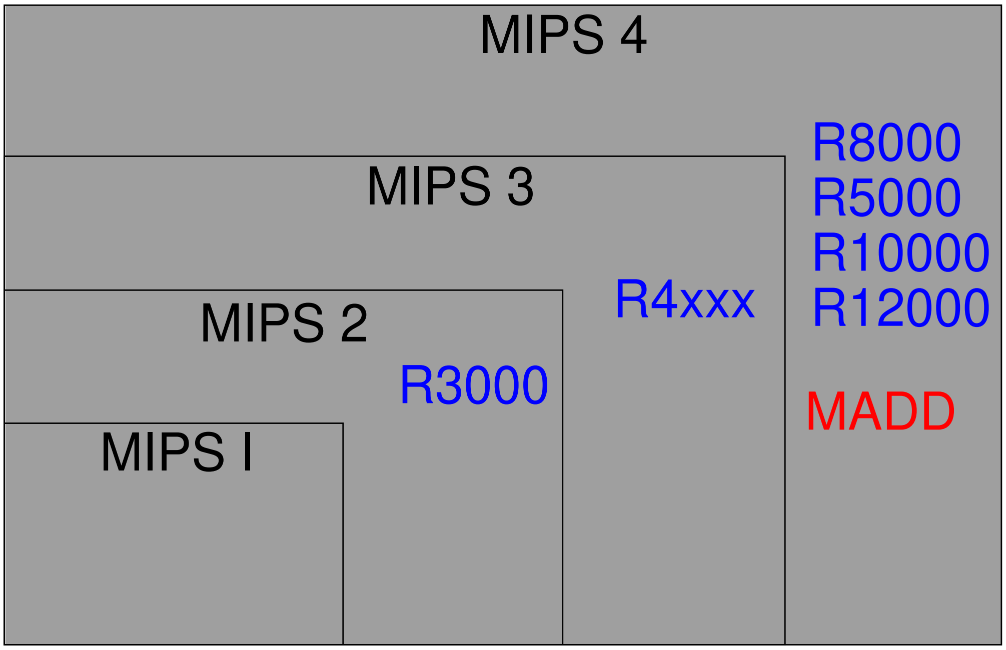
MIPS App Binary Interface (ABI)

4

C type	32–bit		LP64	F77 type
char	8		8	character
short int	16		16	integer*2
int	32		32	integer
long int	32	→	64	–––
long long int	64		64	integer*8
pointer	32	→	64	pointer
float	32		32	real
double	64		64	real*8
long double	64	→	128	real*16

Instruction Set Arquitecture (ISA)

5



Conjunto de instrucciones ensamblador soportadas
para cada tipo de arquitectura

Target Processor

6

Target Processor

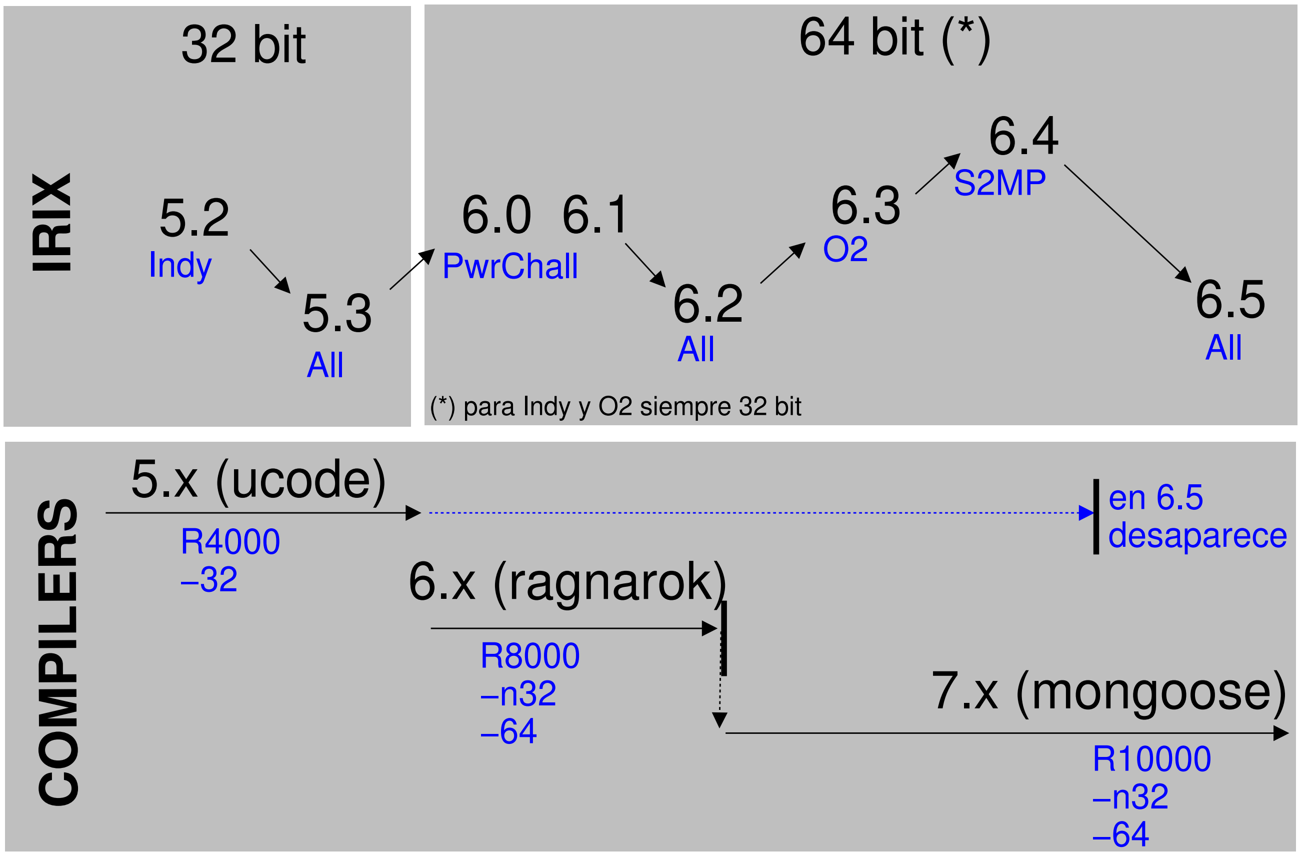
–r4000
–r5000
–r8000
–r10000

especifica cúal es el mejor scheduling de instrucciones en función del procesador para el cual queremos optimizar.

En la fase de link, especifica cúales son las librerías más apropiadas para linkar si es que existen diversas versiones optimizadas para distintas arquitecturas.

7

Compiladores: ucode, ragnarok y mongoose.



8

default compilation mode, supported platforms, compilation capabilities

5.3

def. compil mode: –32 –mips1

sup platf: R3K, R4K

compil. capabil:
–32 –mips1 (ucode)
–32 –mips2 (ucode)

6.2

def. compil mode: R4K, R5K, R10K: –32 –mips2
R8K: –64 –mips4

sup platf: R4K, R5K, R8K y R10K

compil. capabil:
–32 –mips1 (ucode)
–32 –mips2 (ucode)
–n32 –mips3 (ragnarok)
–n32 –mips4 (ragnarok)
–64 –mips3 (ragnarok)
–64 –mips4 (ragnarok)

7.1

def. compil mode: R4K, R5K, R10K: –32 –mips2
R8K: –64 –mips4

sup platf: R4K, R5K, R8K y R10K

compil. capabil:
–32 –mips1 (ucode)
–32 –mips2 (ucode)
–n32 –mips3 (mongoose)
–n32 –mips4 (mongoose)
–64 –mips3 (mongoose)
–64 –mips4 (mongoose)

7.2

def. compil mode: /etc/compiler.defaults

```
atrill@itv> cat /etc/compiler.defaults
–DEFAULT:abi=n32:isa=mips4:proc=r10k
```

sup platf: R4K, R5K, R8K y R10K

compil. capabil:
–32 –mips1 (ucode)
–32 –mips2 (ucode)
–n32 –mips3 (mongoose)
–n32 –mips4 (mongoose)
–64 –mips3 (mongoose)
–64 –mips4 (mongoose)

9

Compiladores v7.2 defaults

- Defaults varian según plataformas & OS:
  - IRIX 6.2 & 6.3: –O0 –32 –mips2
  - Challenge, Indy, and O2: –O0 –32 –mips2
  - PowerChallenge: –O0 –n32 –mips4 –r8k
  - Origin and Octane: –O0 –n32 –mips4 –r10k
  - CrayOrigin: –O2 –n32 –mips4 –OPT:ieee_ar=3
  - PowerChallenge: –O0 –n32 –mips4 –r8k

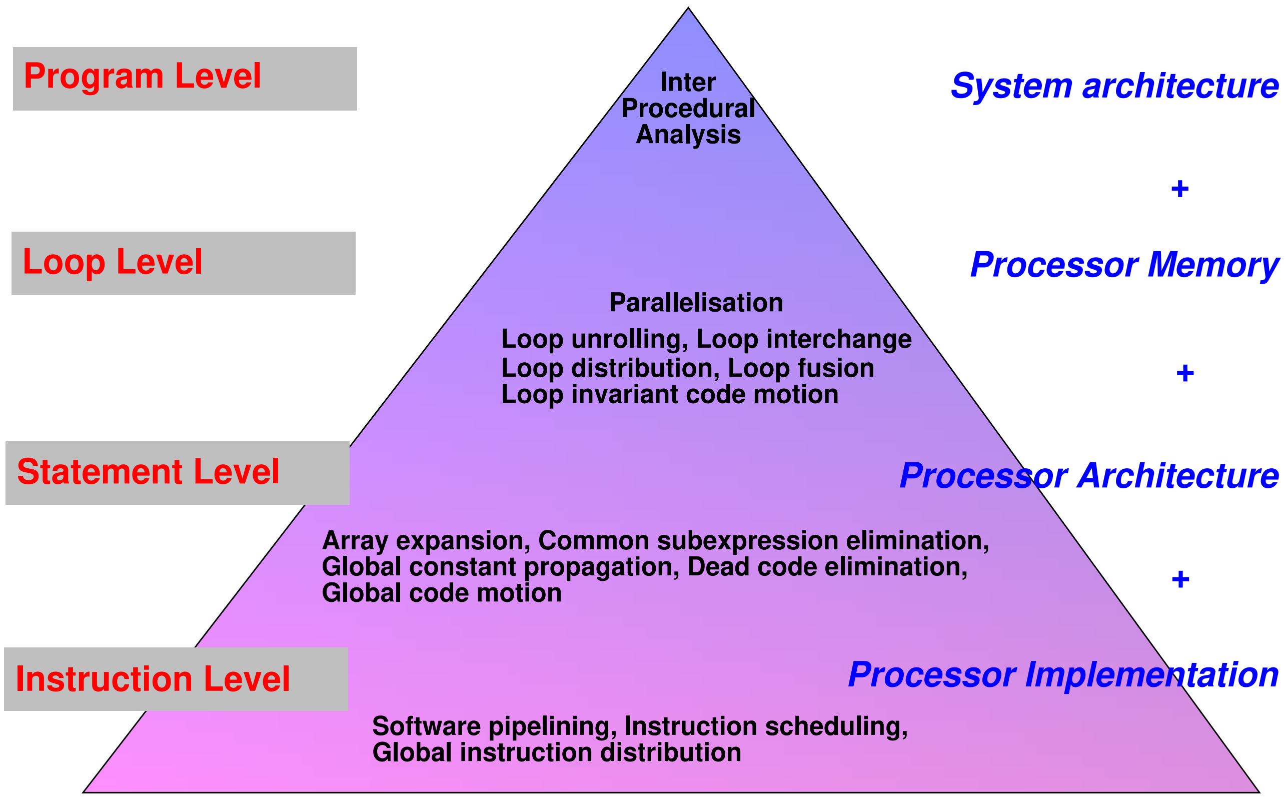
10

IRIX 6.4 execution/compilation table

	–32		–n32		–64	
	MIPS 1	MIPS 2 –32	MIPS 3 –n32 –mips3	MIPS 4 –n32 –mips4	MIPS 3 –64 –mips3	MIPS 4 –64 –mips4
Indigo, Indigo2(R4k),Indy(R4k),Challenge S(R4k)						
Ejecución	✓	✓	✓			
Desarrollo		✓	✓	✓	✓	✓
Indy(R5k),Challenge S(R5k)						
Ejecución	✓	✓	✓	✓		
Desarrollo		✓	✓	✓	✓	✓
Challenge Dm,L,XL(R4k), Onyx(R4k)						
Ejecución	✓	✓	✓		✓	
Desarrollo		✓	✓	✓	✓	✓
PwrIndigo2, PowerChall, PwrOnyx, & all R10k systems						
Ejecución	✓	✓	✓	✓	✓	✓
Desarrollo		✓	✓	✓	✓	✓
O2 (R5k y R10k)						
Ejecución	✓	✓	✓	✓		
Desarrollo		✓	✓	✓	✓	✓

MIPSpro Compiler Overview

11



Compiler Optimizations(MPro)

12

Architecture Specific
- software pipelining
- instruction scheduling
- automatic blocking
- register blocking
- array padding
- global instruction distribution

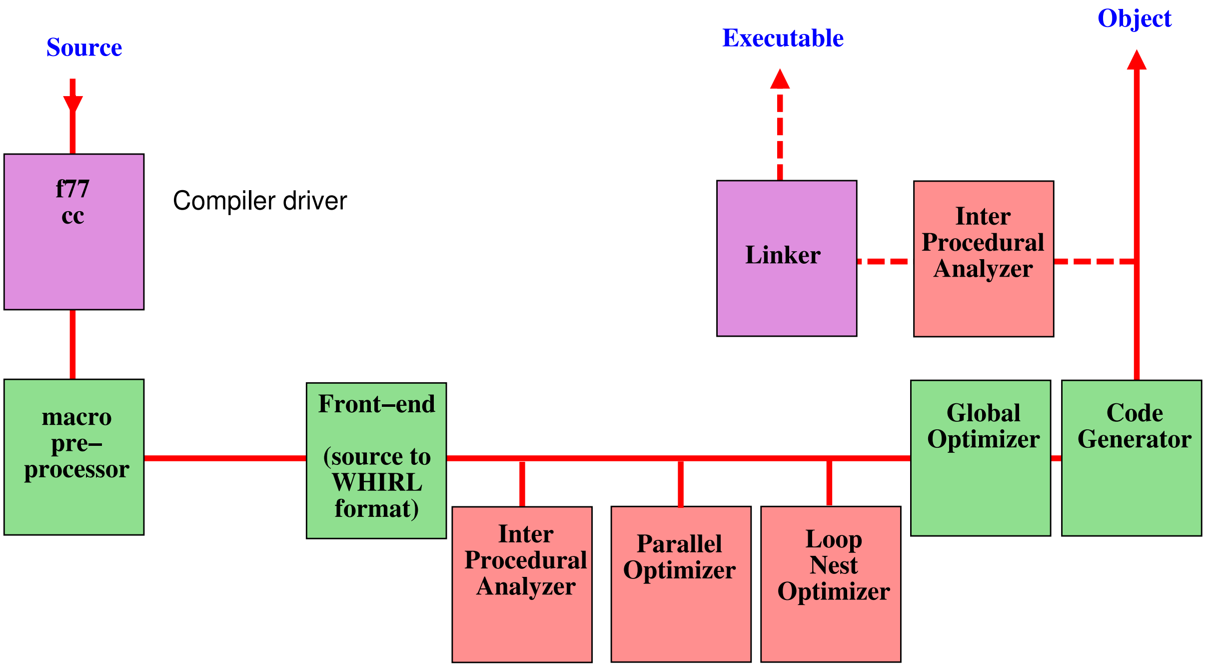
Loop Level
- loop unrolling
- loop interchange
- unroll–and–jam
- loop distribution
- loop fusion
- loop invariant code motion
- sum reduction
- parallelization

Statement Level
- array expansion
- common subexpression elimination
- global constant propagation
- dead code elimination
- global code motion

Procedure Level
- procedure inlining
- interprocedural analysis

WHIRL model
(Winning Hierarchical Intermediate Representation Language)

1 3



WHIRL model: Compiler Concepts

1 4

WHIRL – A performance chasers delight. It is the intermediate representation of your program on which the compiler operates and is the backbone of the high performance mongoose compiler.

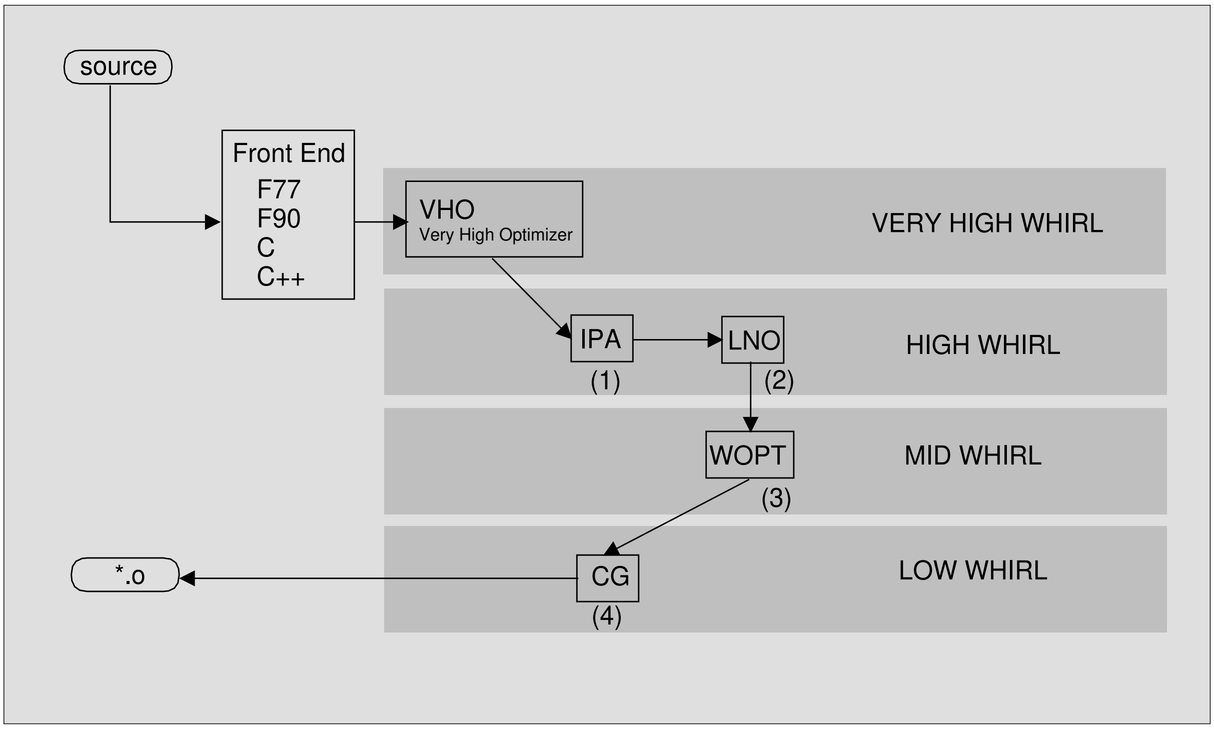
Inter Procedural Analyzer (IPA) – Optimize your entire program. This works across procedure and file boundaries. It performs Interprocedural constant propagation, inter file inlining, aliasing/address analysis and more.

Loop Nest Optimizer (LNO) – Optimize the memory performance of the loop nests in your program. This phase is responsible for dependence analysis, register/cache blocking, loop fission/fusion, automatic prefetching, and array padding.

Global Optimizer – Significantly enhanced from the days of the Ucode optimizer. Performs all the classic optimizations and more such as recovering do loops from while loops, identifying register variables.

Code Generator – The true backend of the compiler. Is responsible for optimizations such as software pipelining, inner loop unrolling, and scheduling to utilize ILP (Instruction Level Parallelism). Also allocates registers using both global and local register allocation.

WHIRL levels



Compiler flags y groups

–OPT:*key=val*	Class of additional optimisations
–TARG:*key=val*	Control target architecture
–TENV:*key=val*	Control target environment
–FLIST/CLIST:*key=val*	F77/C listing control
–LIST:*key=val*	Options to control the listing
–INLINE:*key=val*	Inliner control
–IPA:*key=val*	Inter Procedural Analyzer control
–LNO:*key=val*	Loop Nest Optimizer control

In general multiple {key=val} combinations are colon separated:

```
–OPT:roundoff=2:alias=restrict
```

Note: La excepción que confirma la regla: los flags de –INLINE, que van separados por comas

Compiler Optimizations Levels

17

–O0

Default.
No optimization (usually used for debugging i.e. –g).

–O1

algebraic simplification
common subexpression elimination
constant propagation & folding
dead–code elimination
invariant IF floating
loop unrolling & rerolling
loop fusion & peeling
array expansion

Compiler Optimizations(MPro)

18

–O2

all –O1 optimizations
loop blocking & interchange
global constant propagation
basic block fusion
strength reduction
induction variable simplification
fenceposting
backward motion of code invariants
forward motion of stores
collapsing IF
global copy propagation
arithmetic expression folding
global code motion

–O3

all –O2 optimizations
software pipelining
floating point optimizations

Loop Unrolling

```
do i=1,N
   a(i) = a(i) + s * b(i)
enddo
```

```
do i=1,N,2
   a(i  ) = a(i  ) + s * b(i  )
   a(i+1) = a(i+1) + s * b(i+1)
enddo
do i=i,N
   a(i) = a(i) + s * b(i)
enddo
```

* Reduce importance of loop control operations

* May augment cache reuse