

A Quantitative Analysis of Aspects in the eCos Kernel

Daniel Lohmann

Fabian Scheler

Reinhard Tartler

Olaf Spinczyk

Wolfgang Schröder-Preikschat

Department of Computer Science IV
Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

<http://www4.cs.fau.de/>



Motivation: Cross-cutting Concerns

Cross-cutting Concern: A concern, whose implementation is scattered over the implementation of other concerns

- Enforcement of global policies and strategies
 - synchronization
 - instrumentation
 - tracing/logging
 - protection
 - ...

- Enforcement of fine-grained configuration options
 - optional features
 - alternative implementations
 - ...

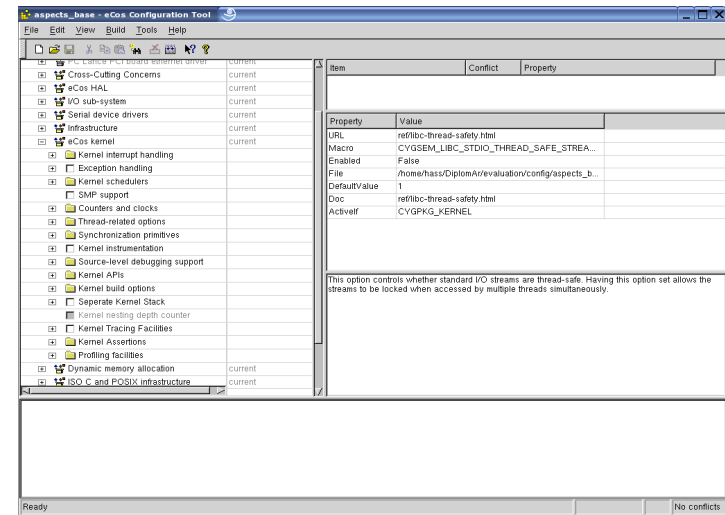


eCos: An OS Product Line

- Open source OS product line for embedded applications
 - developed and maintained by RedHat
 - supports a high number of 16/32 bit architectures
 - kernel written in C++

- Goal: static configurability
 - 63 selectable packages
 - 761 selectable configuration options

- Configuration approach
 - package selection (coarse-grained configuration)
 - **conditional compilation** (fine-grained configuration)

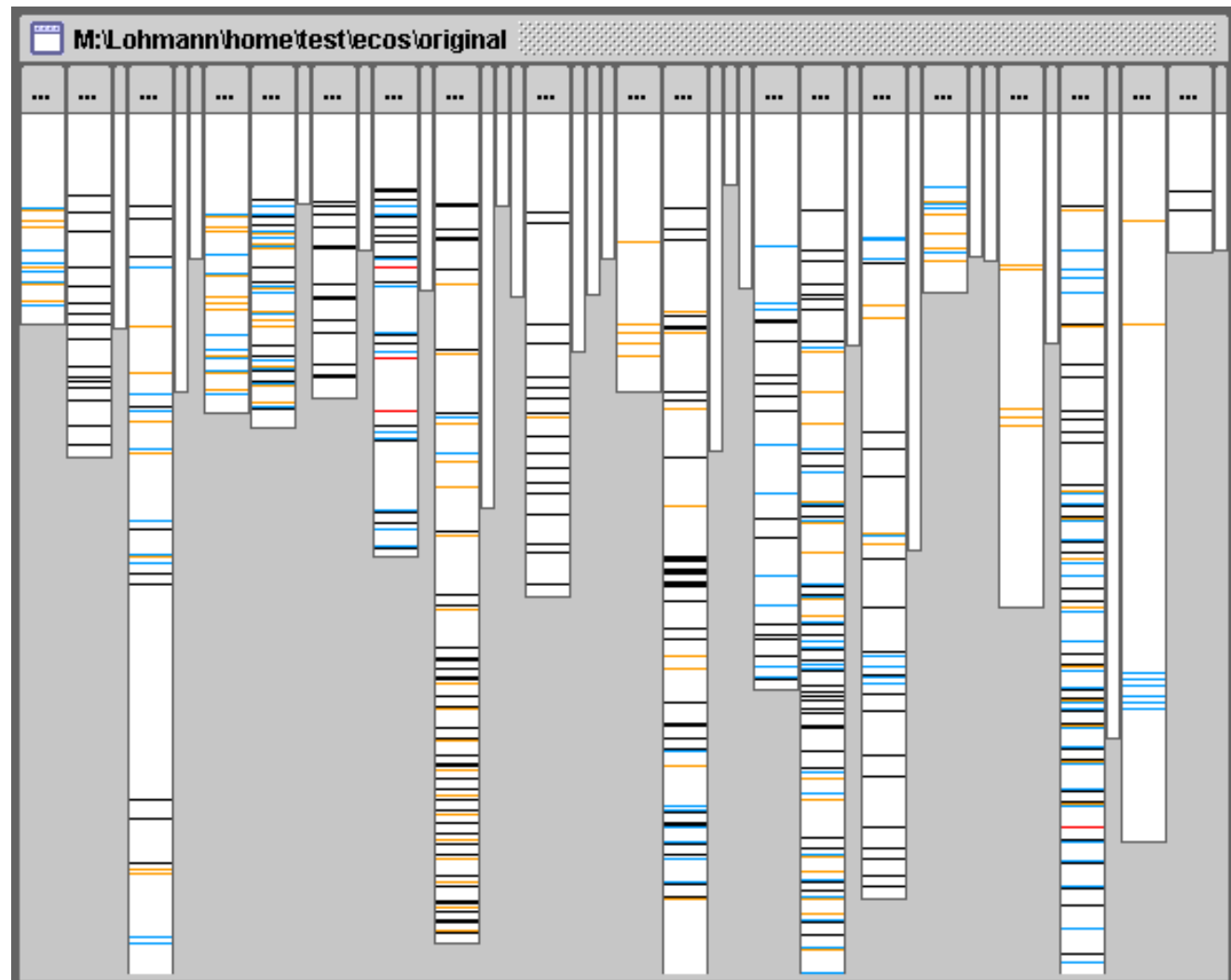


eCos: Enforcement of Some Global Policies

synchronization

instrumentation

tracing



eCos: Distribution of a Configuration Option

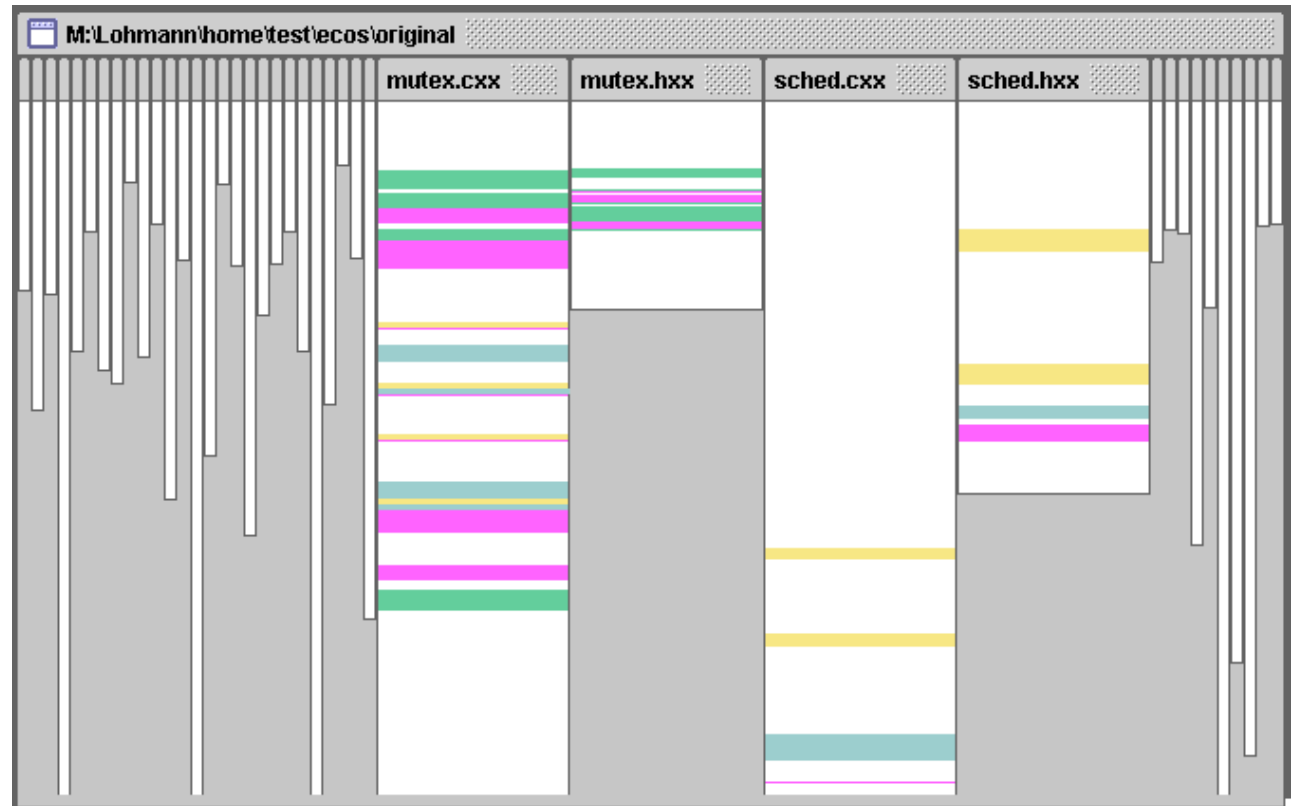
Variants of the optional mutex priority inversion protocol

simple

ceiling

inheritance

dynamic



eCos: Implementation Example

27 lines of code

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner     = NULL;
#if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}
```



eCos: Implementation Example

2 lines for the tracing policy

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner     = NULL;
#if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}
```



eCos: Implementation Example

21 (almost unreadable) lines for optional features

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner     = NULL;
#if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
    defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
    protocol = INHERIT;
#endif
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
    protocol = CEILING;
    ceiling  = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
#endif
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
    protocol = NONE;
#endif
#else // not (DYNAMIC and DEFAULT defined)
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
#ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
    // if there is a default priority ceiling defined, use that to initialize
    // the ceiling.
    ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
#else
    ceiling = 0; // Otherwise set it to zero.
#endif
#endif
#endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}
```



eCos: Implementation Example

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner    = NULL;
    #if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
        defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
        protocol = INHERIT;
    #endif
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_CEILING
        protocol = CEILING;
        ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
    #endif
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_NONE
        protocol = NONE;
    #endif
    #else // not (DYNAMIC and DEFAULT defined)
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY
        // if there is a default priority ceiling defined, use that to initialize
        // the ceiling.
        ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
    #else
        ceiling = 0; // Otherwise set it to zero.
    #endif
    #endif
    #endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}
```

4 lines for the
basic implementation



eCos: Implementation Example

```
Cyg_Mutex::Cyg_Mutex() {
    CYG_REPORT_FUNCTION();
    locked    = false;
    owner    = NULL;
    #if defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT) && \
        defined(CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC)
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_INHERIT
        protocol = INHERIT;
    #endif
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_CEILING
        protocol = CEILING;
        ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRI;
    #endif
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_ONE
        protocol = ONE;
    #endif
    #else // not (DYNAMIC and DEFAULT) defined
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DYNAMIC
    #ifndef CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT
        // if there is a ceiling
        // the ceiling is the priority
        ceiling = CYGSEM_KERNEL_SYNCH_MUTEX_PRIORITY_INVERSION_PROTOCOL_DEFAULT_PRIORITY;
    #endif
    #else
        ceiling = 0;
    #endif
    #endif
    #endif // DYNAMIC and DEFAULT defined
    CYG_REPORT_RETURN();
}
```

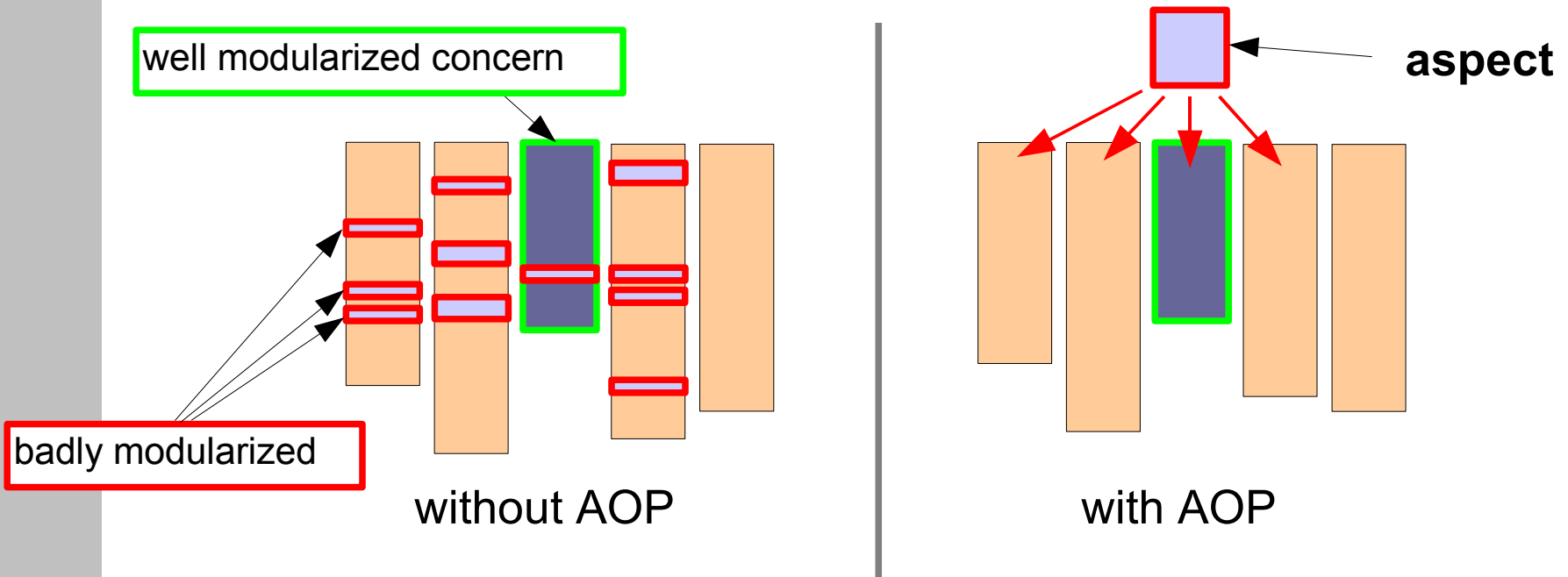
Well, ...

- comprehensability ?
- extensability ?
- reuseability ?
- maintainability?



Aspect-Oriented Programming

AOP provides language means to encapsulate cross-cutting and scattered concerns



Aspects: The Basic Idea

Seperation of *what* from *where*

- **join-points** (*where*)
 - positions in the **static structure** or **dynamic control flow** (event)
 - given **declaratively** by **pointcut expressions**
- **advice** (*what*)
 - additional **elements** (members, ...) to introduce at certain join-points of the static structure (classes, structs)
 - additional **behaviour** (code to execute) to superimpose
 - **before**, **after**
 - **around** (instead of)
 - certain join-points of the dynamic control flow



Example: Priority Ceiling Protocol



```
aspect priority_ceiling {  
  
    void call_clear_ceiling(Cyg_Thread*);  
    ...  
  
    advice "Cyg_Mutex" : cyg_priority ceiling;  
    ...  
  
    advice construction("Cyg_Mutex") : after() {  
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;  
    }  
  
    advice call("% Cyg_Mutex::lock_inner(...)")  
        && within("% Cyg_Mutex::lock(...)")  
        && args(self)  
        : after(Cyg_Thread* self)  
    {  
        if(!(*tjp->result())) {  
            call_clear_ceiling(self);  
        }  
    }  
    ...  
};
```

what **where**

Example: Priority Ceiling Protocol



```
aspect priority_ceiling {  
  
    void call_clear_ceiling(Cyg_Thread*);  
    ...  
    advice "Cyg_Mutex" : cyg_priority ceiling;  
    ...  
  
    advice construction("Cyg_Mutex") : after() {  
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;  
    }  
  
    advice call("% Cyg_Mutex::lock_inner(...)")  
        && within("% Cyg_Mutex::lock(...)")  
        && args(self)  
        : after(Cyg_Thread* self)  
    {  
        if(!(*tjp->result())) {  
            call_clear_ceiling(self);  
        }  
    }  
    ...  
};
```

what

where

Introduce a data member *ceiling* into all classes named *Cyg_Mutex*

Example: Priority Ceiling Protocol



```
aspect priority_ceiling {  
  
    void call_clear_ceiling(Cyg_Thread*);  
    ...  
  
    advice "Cyg_Mutex" : cyg_priority ceiling;  
    ...  
  
    advice construction("Cyg_Mutex") : after() {  
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;  
    }  
  
    advice call("% Cyg_Mutex::lock_inner(...)")  
        && within("% Cyg_Mutex::lock(...)")  
        && args(self)  
        : after(Cyg_Thread* self)  
    {  
        if(!(*tjp->result())) {  
            call_clear_ceiling(self);  
        }  
    }  
    ...  
};
```

what

where

Execute *initialization code*
after the **construction** of a
Cyg_Mutex instance

Example: Priority Ceiling Protocol



```
aspect priority_ceiling {  
  
    void call_clear_ceiling(Cyg_Thread*);  
    ...  
  
    advice "Cyg_Mutex" : cyg_priority ceiling;  
    ...  
  
    advice construction("Cyg_Mutex") : after() {  
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;  
    }  
  
    advice call("% Cyg_Mutex::lock_inner(...)")  
        && within("% Cyg_Mutex::lock(...)")  
        && args(self)  
        : after(Cyg_Thread* self)  
    {  
        if(!(*tjp->result())) {  
            call_clear_ceiling(self);  
        }  
    }  
    ...  
};
```

what **where**

After a **call** to any overload of *Cyg_Mutex::lock_inner* that occurs...

Example: Priority Ceiling Protocol



```
aspect priority_ceiling {  
  
    void call_clear_ceiling(Cyg_Thread*);  
    ...  
  
    advice "Cyg_Mutex" : cyg_priority ceiling;  
    ...  
  
    advice construction("Cyg_Mutex") : after() {  
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;  
    }  
  
    advice call("% Cyg_Mutex::lock_inner(...)")  
        && within("% Cyg_Mutex::lock(...)")  
        && args(self)  
        : after(Cyg_Thread* self)  
    {  
        if(!(*tjp->result())) {  
            call_clear_ceiling(self);  
        }  
    }  
    ...  
};
```

what **where**

After a **call** to any overload of *Cyg_Mutex::lock_inner* that occurs...

...while being **within** *Cyg_Mutex::lock* and takes...

Example: Priority Ceiling Protocol



```
aspect priority_ceiling {  
  
    void call_clear_ceiling(Cyg_Thread*);  
    ...  
  
    advice "Cyg_Mutex" : cyg_priority ceiling;  
    ...  
  
    advice construction("Cyg_Mutex") : after() {  
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;  
    }  
  
    advice call("% Cyg_Mutex::lock_inner(...)")  
        && within("% Cyg_Mutex::lock(...)")  
        && args(self)  
        : after(Cyg_Thread* self)  
    {  
        if(!(*tjp->result())) {  
            call_clear_ceiling(self);  
        }  
    }  
    ...  
};
```

what **where**

After a **call** to any overload of *Cyg_Mutex::lock_inner* that occurs...

...while being **within** *Cyg_Mutex::lock* and takes...

...an **argument** of type *Cyg_Thread**...

Example: Priority Ceiling Protocol



```
aspect priority_ceiling {  
  
    void call_clear_ceiling(Cyg_Thread*);  
    ...  
  
    advice "Cyg_Mutex" : cyg_priority ceiling;  
    ...  
  
    advice construction("Cyg_Mutex") : after() {  
        tjp->that()->ceiling = CYGSEM_DEFAULT_PRIORITY;  
    }  
  
    advice call("% Cyg_Mutex::lock_inner(...)")  
        && within("% Cyg_Mutex::lock(...)")  
        && args(self)  
        : after(Cyg_Thread* self)  
    {  
        if(!(*tjp->result())) {  
            call_clear_ceiling(self);  
        }  
    }  
    ...  
};
```

what

where

After a **call** to any overload of *Cyg_Mutex::lock_inner* that occurs...

...while being **within** *Cyg_Mutex::lock* and takes...

...an **argument** of type *Cyg_Thread**...

...check the result value and clear ceiling, if required

Summary: AOP

- AOP provides additional means for separation of concerns
 - declarative match mechanism
 - join-points denote events/positions with specific **semantics**
 - **it's not just patching!**
- Does it really help?
 - several publications say so
 - separating prefetching code in FreeBSD (Coady, 2001)
 - integrating the Bossa scheduler into Linux (Åberg, 2003)
 - ... “...*improved* evolvability / comprehensability / configurability..”
 - counter-examples have been published as well
 - separating network code optimizations (Siadat, 2006)
“...comprehensability was *not improved.*”
- More **experience** and **larger studies** required!



Summary: AOP

- AOP provides additional means for separation of concerns
 - declarative match mechanism
 - join-points denote events/positions with specific **semantics** ...
 - **it's not just patching!**
- Does it cost?
 - **separation of concerns**
 - separation of concerns
 - it's not just patching!
 - ...
 - counter-examples have been published as well
 - separating network code optimizations (*Siadat, 2006*)
“...comprehensability was **not improved.**”
- More **experience** and **larger studies** required!



Quantifying the Overhead of AOP

Target: AspectC++ language and weaver

- open source aspect weaver for C++
- transforms AspectC++ code into C/C++ code
- platform/compiler-independent



<http://www.aspectc.org>

Conducted:

1. A series of μ -Benchmarks for AspectC++ constructs
2. Larger comparative study by refactoring eCos



1. Costs of AspectC++ language features

a) incrementer

		advice		cycles		stack		code	
		abs	Δ	abs	Δ	abs	Δ	abs	Δ
execution	<i>tangled</i>	4		0		4128			
	before	6	2	0	0	4128	0		
	after	6	2	0	0	4128	0		
	around	6	2	0	0	4128	0		
call	before	6	2	0	0	4128	0		
	after	6	2	0	0	4128	0		
	around	6	2	0	0	4128	0		

b) multiaspect

		# aspect	cycles		stack		code	
			abs	Δ	abs	Δ	abs	Δ
execution	1	6		0		4080		
	2	5	-1	0	0	4080	0	
	3	6	1	0	0	4096	16	
call	1	6		0		4096		
	2	5	-1	0	0	4096	0	
	3	6	1	0	0	4096	0	

c) parameters, jp-api

		cycles		stack		code	
		abs	Δ	abs	Δ	abs	Δ
int B::bar(...)	tjp->						
	<i>plain</i> , n=0	5		16		3968	
	that()	7	2	20	4	3968	0
	target()	8	3	20	4	3968	0
	result()	11	6	16	0	3968	0
<i>plain</i> , n=1		13		24		3968	
	arg<0>()	13	0	24	0	3968	0
	<i>plain</i> , n=2	13		32		3984	
arg<1>()	13	0	32	0	3984	0	

d) dynamic pointcuts

pointcut function	cycles	stack	code	data
	Δ	Δ	Δ	Δ
cflow()				4
enter/leave	6	16	8	
test	12	52	56	
that()	10	24	128	50
target()	10	24	144	50



1. Costs of AspectC++ language features

a) incrementer

		cycles		stack		code	
		abs	Δ	abs	Δ	abs	Δ
execution	<i>tangled</i>	4		0		4128	
	before	6	2	0	0	4128	0
	after	6	2	0	0	4128	0
	around	6	2	0	0	4128	0
call	before	6	2	0	0	4128	0
	after	6	2	0	0	4128	0
	around	6	2	0	0	4128	0

b) multiaspect

		# aspect	cycles		stack		code	
			abs	Δ	abs	Δ	abs	Δ
execution	1	6		0		4080		
	2	5		0	0	4080	0	
	3	6	1	0	0	4096	16	
call	1	6		0		4096		
	2	5	-1	0	0	4096	0	
	3	6	1	0	0	4096	0	

c) parameters, jp-api

		cycles		stack		code	
		abs	Δ	abs	Δ	abs	Δ
tjp->	<i>plain, n=0</i>	5		16		3968	
	that()	7	2	20	4	3968	0
	target()	8		20	4	3968	0
	result()	11	5	16	0	3968	0
	<i>plain, n=1</i>	5		24		3968	
int B::bar(...)	arg<0>	13	0	24	0	3968	0
	<i>plain, n=2</i>	13		32		3984	
	arg<1>()	13	0	32	0	3984	0

d) dynamic pointcuts

pointcut function	cycles	stack	code	data
	Δ	Δ	Δ	Δ
cflow()				4
enter/leave	6	16	8	
test	12	52	56	
that()	10	24	128	50
target()	10	24	144	50

Please find details in the paper!



1. Costs of AspectC++ language features

a) incrementer

advice	cycles		stack		code	
	abs	Δ	abs	Δ	abs	Δ
<i>tangled</i>	4		0		4128	

b) multiaspect

# aspect	cycles		stack		code	
	abs	Δ	abs	Δ	abs	Δ

Results

- simple *before/after/around* advice for *call/execution* join-points does **not induce costs**
- accessing *join-point context* (arg, result, ...) may **induce** some stack and CPU **costs** – but very low
- *dynamic pointcut functions* (cflow, that, target) **induce noticeable overhead**

<i>plain, n=2</i>	13		32		3984	
<i>arg<1>()</i>	13	0	32	0	3984	0

<i>target()</i>	10	24	144	50
-----------------	----	----	-----	----



2. Comparative Study with eCos

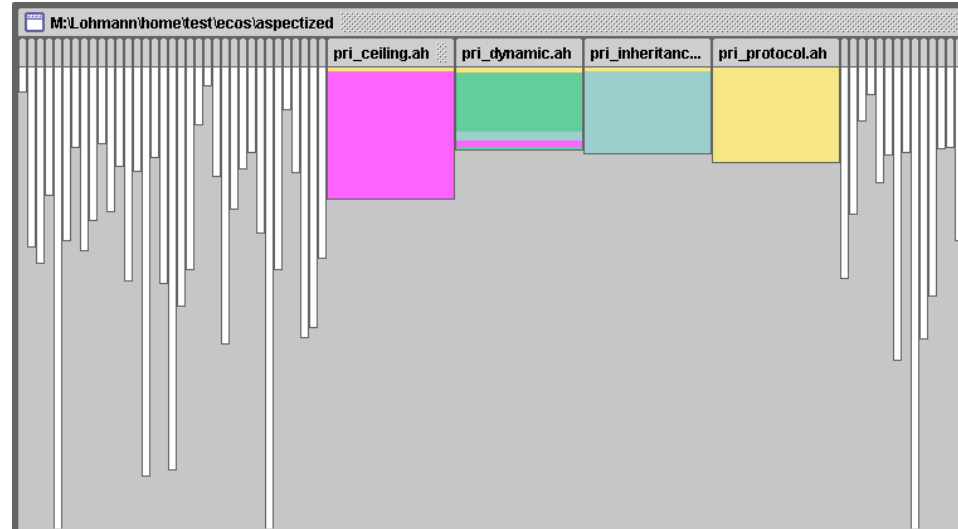
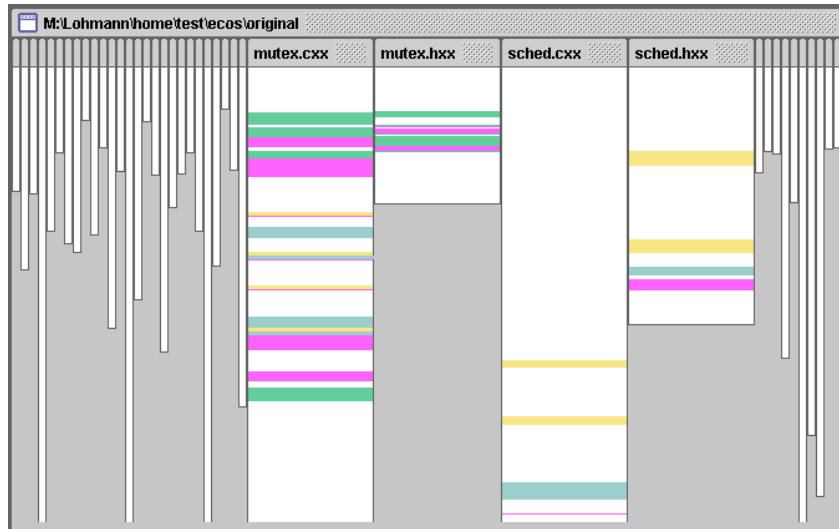
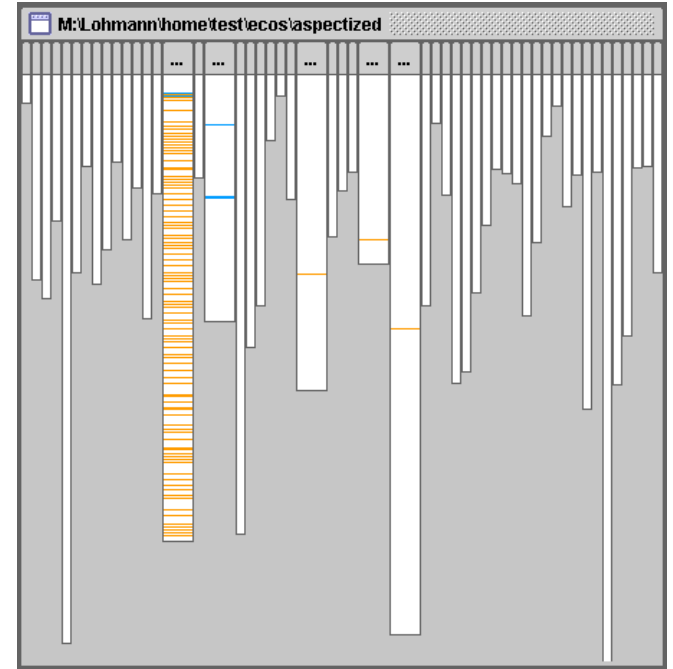
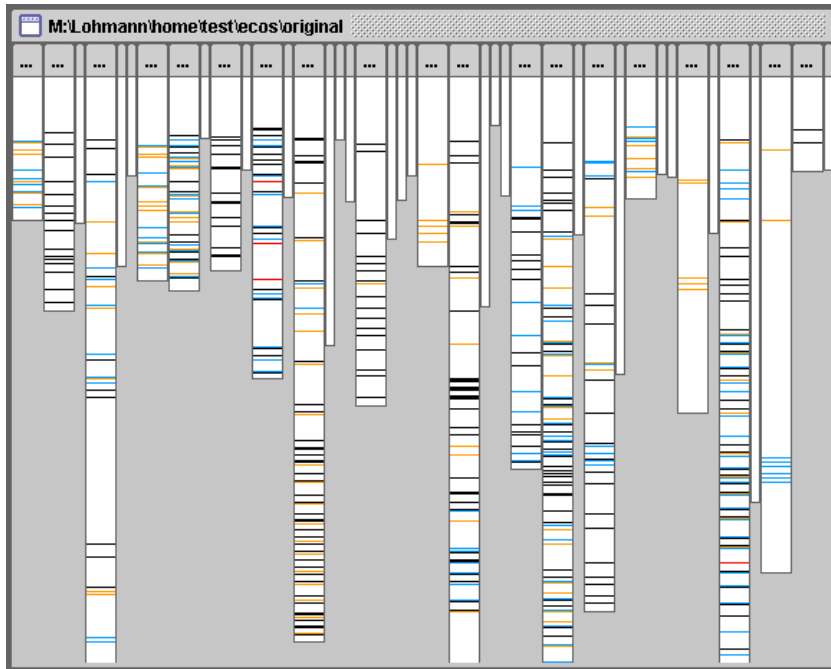
- **Refactored:** *original kernel* → *aspectized kernel*
 - **3 cross-cutting policies**
 - interrupt synchronization 187 invocations → 160 code join-points
 - kernel instrumentation 162 invocations → 139 code join-points
 - tracing 336 invocations → 632 code join-points
 - **12 configuration options**
 - mutex features
 - thread features
- **Compared:** *original kernel* ↔ *aspectized kernel*
 - scattering
 - performance
 - memory footprint



original kernel



aspectized kernel



Evaluation: Performance and Memory Costs

- Evaluation based on 3 small multi-threaded test programs
 - specifically use affected kernel functions
 - overhead, if any, should become evident with this setup
 - each measured with 13 different eCos configurations

a) thread cyg_thread_...		b) mutex cyg_mutex_...		c) semaphore cyg_semaphore_...		
1	..._create()	215	..._init()	52	..._init()	62
2	..._resume()	327	..._lock()	47	..._post()	50
3	..._resume()	127	..._unlock()	22	..._wait()	723
4	..._yield()	274	..._try_lock()	46	..._trywait()	416
5	..._exit()	354	..._try_lock()	49	..._trywait()	44
6	..._yield()	77	..._lock()	381	..._wait	46
7	..._resume()	91	..._unlock()	429	..._post()	22
8	..._kill()	102	..._destroy()	17	..._destroy()	19
9	..._suspend()	336				
10	..._suspend()	96				
11	..._suspend()	53				
12	..._resume()	63				
13	..._resume()	115				
14	..._delete()	38				
Σ	[cycles]	2268	[cycles]	1043	[cycles]	1382

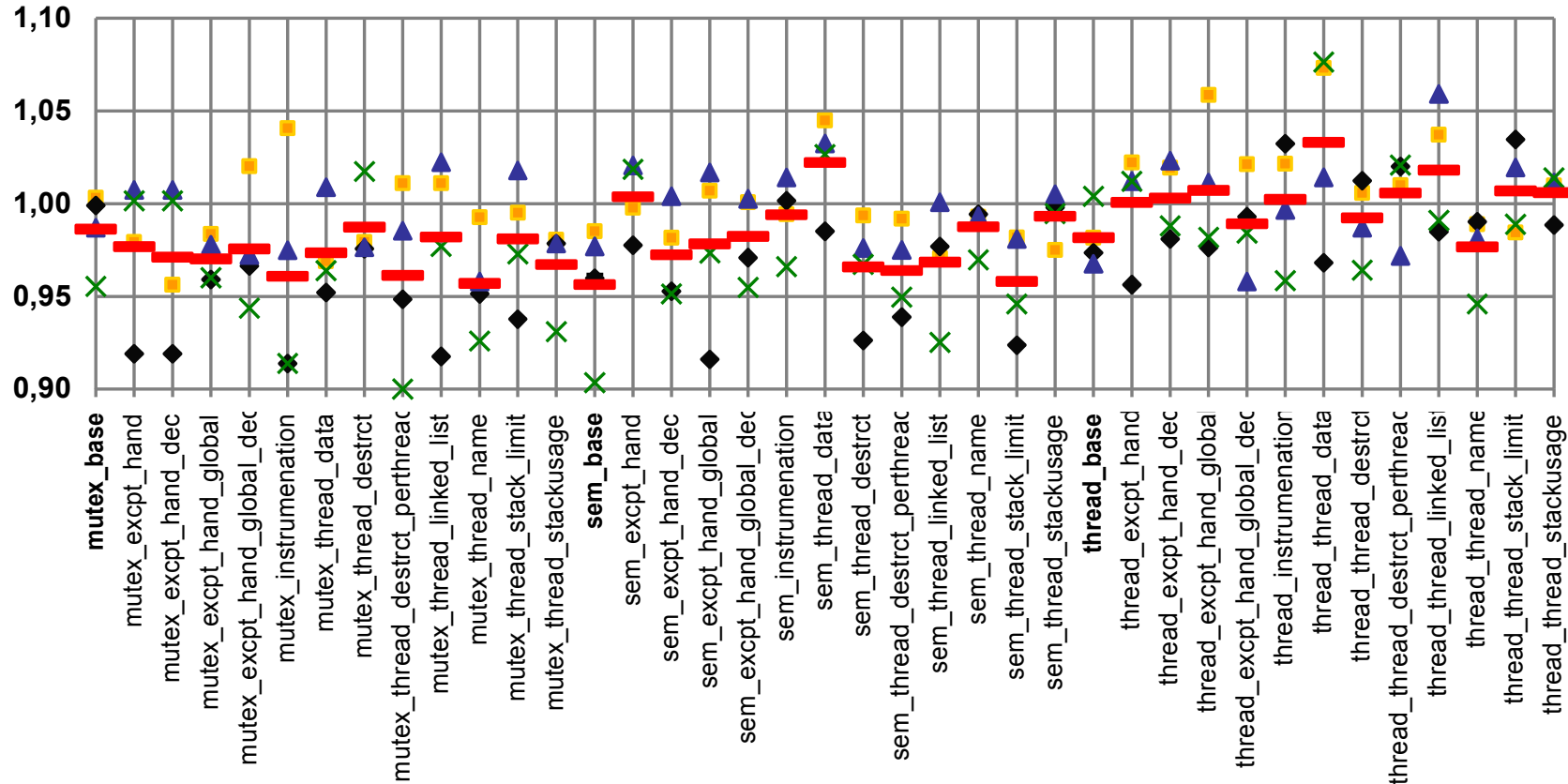


Evaluation: Performance Costs

cost factor

aop/plain [cycles]

■ Pentium ◆ P3 ▲ Athlon × P4 - [mean]



3 Testcases
a 13 configurations

mutex

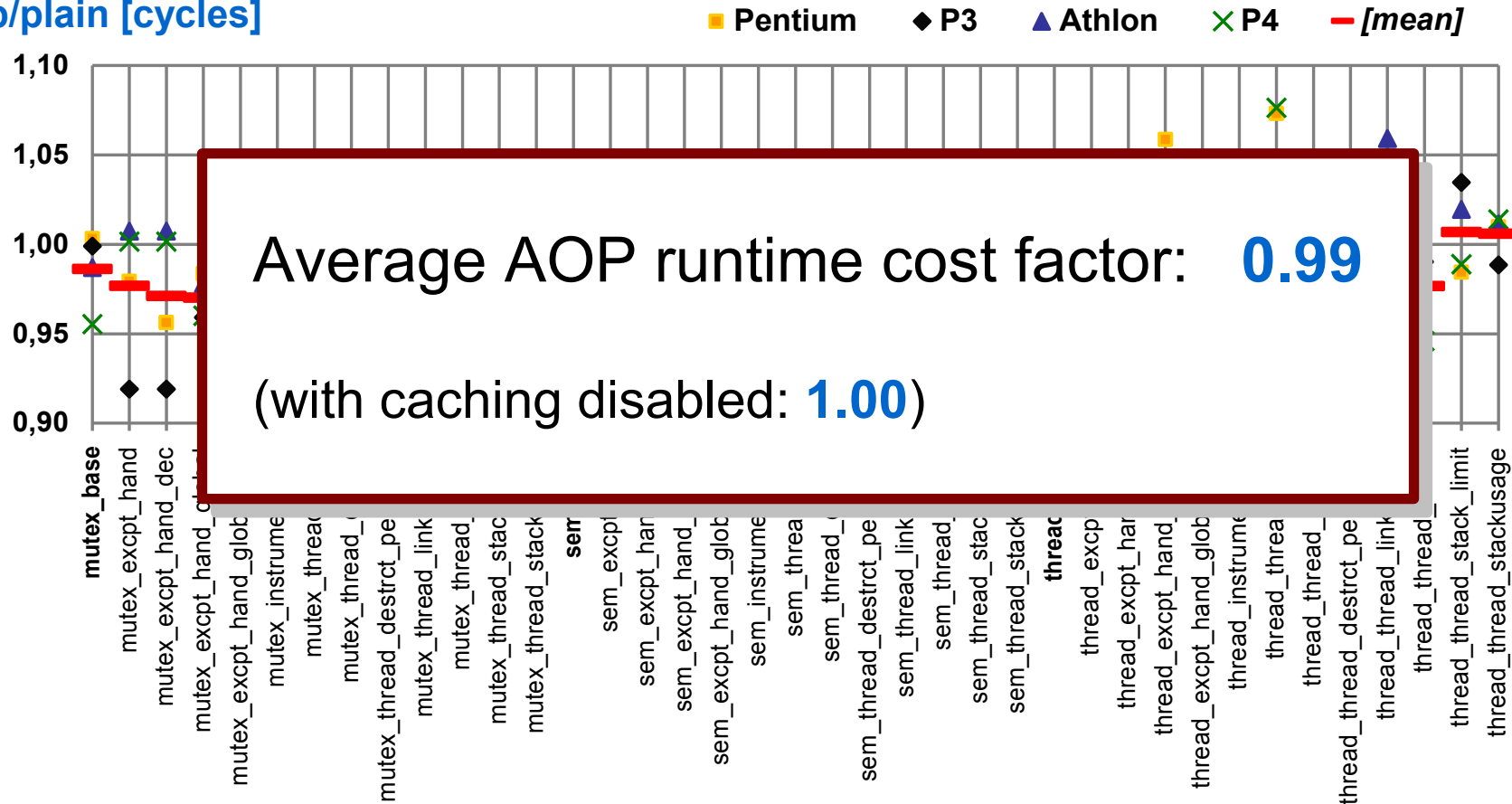
semaphore

thread

Evaluation: Performance Costs

cost factor

aop/plain [cycles]



Average AOP runtime cost factor: **0.99**

(with caching disabled: **1.00**)

3 Testcases
a 13 configurations

mutex

semaphore

thread

Evaluation: Memory Costs

cost factor
aop/plain [bytes]

	[bytes]	ROM		RAM		stack	
		Δ	%	Δ	%	Δ	%
01	<i>mutex_base</i>	101	0,6	0	0,0	24	2,0
02	<i>mutex_excpt_hand</i>	55	0,3	0	0,0	24	2,0
03	<i>mutex_excpt_hand_dec</i>	-35	-0,2	0	0,0	24	2,0
04	<i>mutex_excpt_hand_global</i>	92	0,5	0	0,0	24	2,0
05	<i>mutex_excpt_hand_global_dec</i>	85	0,5	0	0,0	24	2,0
06	<i>mutex_instrumentation</i>	543	3,0	0	0,0	4	0,3
07	<i>mutex_thread_data</i>	74	0,4	0	0,0	28	2,3
08	<i>mutex_thread_destrct</i>	74	0,4	0	0,0	28	2,3
09	<i>mutex_thread_destrct_perthread</i>	224	1,3	0	0,0	24	2,0
10	<i>mutex_thread_linked_list</i>	88	0,5	0	0,0	8	0,6
11	<i>mutex_thread_name</i>	146	0,9	0	0,0	16	1,3
12	<i>mutex_thread_stack_limit</i>	311	1,8	0	0,0	24	2,0
13	<i>mutex_thread_stackusage</i>	341	2,0	0	0,0	24	2,0
14	<i>sem_base</i>	91	0,5	0	0,0	24	2,0
15	...	51	0,3	0	0,0	24	2,0
40	Average [%]		0,9		0,0		1,3

Evaluation: Memory Costs

cost factor
aop/plain [bytes]

[bytes]		ROM		RAM		stack	
		Δ	%	Δ	%	Δ	%
01	<i>mutex_base</i>	101	0,6	0	0,0	24	2,0
02	<i>mutex_excpt_hand</i>	55	0,3	0	0,0	24	2,0
03	<i>mutex_excpt_hand_dec</i>	-35	-0,2	0	0,0	24	2,0
04	<i>mutex_excpt_hand_global</i>	92	0,5	0	0,0	24	2,0
05	<i>mutex_excpt_hand_global_dec</i>	85	0,5	0	0,0	24	2,0
06	<i>mutex_instrumentation</i>	543	3,0	0	0,0	4	0,3
07	<i>mutex_thread_data</i>	74	0,4	0	0,0	28	2,3

- Average AOP ROM cost factor: **1.009**
- Average AOP stack cost factor: **1.013**

39	...	57	0,3	0	0,0	27	2,0
40	Average [%]		0,9		0,0		1,3

Lessons Learned

- “Aspectizing” the eCos kernel was relatively easy
 - concerns *conceptually* already separated
 - fine-grained implementation structure offered enough join-points
- Once aspectized, policy-changes could be applied easily
 - scheduler activation points
 - synchronization points
 - design becomes more explicit in the code
- Compile-time resolvable join-points were success factor
 - overhead goes up, if dynamic pointcut functions have to be used
 - see *kernel-stack* extension example in the paper



Conclusions

- Separation of concerns is an issue in system software
 - enforcement of cross-cutting policies
 - fine-grained optional features in product lines
- Aspect-oriented programming is promising
 - provides additional means for separation of concerns
 - declarative concepts to separate *what* from *where*
- **It can be applied cost-neutral**
 - developers can get better separation of concerns for free!
 - well suited for system software



Future Work: The CiAO Project

- **Approach:** Use AOP concepts in OS development from the very beginning
 - if we use aspects from scratch, how far can we get?
 - what are the fundamental design patterns?
- **Goal:** Configurability of **architectural** properties
 - protection (e.g. single address space vs. memory protection)
 - interaction (e.g. procedural vs. message-based)
 - synchronization (e.g. fine-grained vs. coarse-grained)



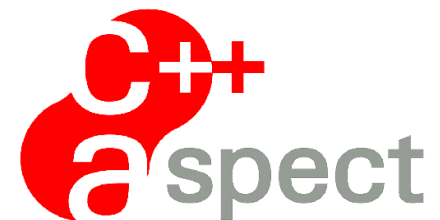
<http://www.aspectc.org>



Future Work: The CiAO Project

- **Approach:** Use AOP concepts in OS development from the very beginning
 - if we use aspects from scratch, how far can we get?
 - what are the fundamental design patterns?
- **Goal:** Configurability of **architectural** properties
 - protection (e.g. single address space vs. memory protection)
 - interaction (e.g. procedural vs. message-based)
 - synchronization (e.g. fine-grained vs. coarse-grained)

**Thank you very much
for your attention**



<http://www.aspectc.org>

