



# Next Generation Operating Environment

Bruce Trask

Director of Software Engineering,  
SDR Products

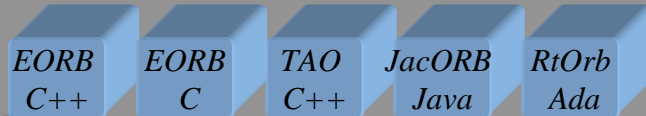
# CORBA Middleware Misconceptions

- ▶ CORBA based Middleware
  - ▶ Too big ...
  - ▶ Too slow ...
  - ▶ Too heavyweight ...
  - ▶ Not designed for small scale systems ...
- ▶ PrismTech Response:
  - ▶ It's possible to run the next generation of CORBA middleware efficiently in even the most resource constrained environment such as a DSP or FPGA

..... and this presentation will demonstrate this !

# CORBA Middleware

## *GPP - Enterprise*



- ▶ Large memory capacity
- ▶ Rich system resource support
- ▶ Operating Systems supports POSIX APIs

## *GPP- Embedded*



- ▶ Limited memory capacity
- ▶ Basic system resources available (tasks, semaphores, mutex, condition variables)
- ▶ Operating Systems may support POSIX APIs

## *DSP*



- ▶ Very constrained memory capacity < 2Mb
- ▶ Basic system resources available (tasks, semaphores, locks, interrupts)
- ▶ Usually no POSIX APIs

## *FPGA*



- ▶ Small amount of logic in FPGA
- ▶ Implemented CORBA features to support SCA
- ▶ Low level transport layer

# Next Generation CORBA on DSP & FPGA

- ▶ DSP platform
  - ▶ Significantly reduced footprint ORBs can fit in on-chip memory.
    - ▶ C ORB deployment < 100k (TI TMS320V5510)
  - ▶ Modular ORB functionality
    - ▶ Pick 'n' Mix approach to building ORB function
  - ▶ Pluggable transport layers allowing smaller and faster transport implementations
    - ▶ ETF based transport TCP, UDP, Shared Memory, Rapid I/O, Raw Ethernet
  - ▶ Pooling of system resources (threads, memory)
  - ▶ System characteristics obeying real-time requirements
- ▶ FPGA platform
  - ▶ ICO an Integrated Circuit ORB



# CORBA on DSP – Key Challenges

## ▶ Size

- ▶ Choice of implementation language and CORBA binding
  - ▶ Apart from direct assembly, C and C++ most common languages used for DSP programming
  - ▶ C and C++ language most commonly supported by compilers
  - ▶ 3<sup>rd</sup> party drivers for devices most commonly available as C libraries

# CORBA on DSP – Size

- ▶ C over C++ ?
  - ▶ C++ can be and is supported
  - ▶ C++ in general produces larger code footprint
  - ▶ Some compilers have difficulty with certain aspects of the C++ language.
  - ▶ C produces smaller code
  - ▶ All language facilities well supported by most C compilers.
  - ▶ C language is very portable

# CORBA on DSP – Size

## ▶ Objects

- ▶ C++ implemented using class, attributes and methods
- ▶ C implemented using struct, members and functions taking struct as argument
- ▶ In general compilers produce more code in C++ than for the equivalent OO approximation in C

## ▶ Exceptions

- ▶ C++ supports exceptions. As such CORBA exceptions usually mapped to native exceptions with often considerable additional size implications
- ▶ C CORBA exception simply implemented as a member in a CORBA::Environment struct passed as an argument to all CORBA function calls

# CORBA on DSP – Size

- ▶ C++ support libraries and runtime
  - ▶ C++ implementations typically make use of generic C++ support libraries (such as STL)
  - ▶ C implementation makes use of customised internal functions for the same functionality

# CORBA on DSP – Size

## ▶ Binding Issues

- ▶ C++ binding more extensive than C with more support operations defined on core and generated classes
- ▶ C++ language binding defines a whole set of additional type support
  - ▶ `_var`, `_mgr`, `_out`
- ▶ C binding uses simpler type mappings.
  - ▶ ‘Any’ a good example.
    - ▶ *C++ : implemented as class with 60 defined methods*
    - ▶ *C: implemented as struct with 3 accessor functions*

# CORBA on DSP - Size

- ▶ Transport
  - ▶ Communication driver implementations can contribute considerably to overall footprint
  - ▶ Example some TCP/IP stack implementations can add 100s of KBs to footprint e.g. BF3Net on TI BIOS consumes 135KB, NDK on TI BIOS consumes 288KB
  - ▶ Choosing a transport implementation wisely will reduce footprint significantly e.g. Link Handler for OSEck on TI DSP consumes approximately 20KB

# CORBA on DSP – Size Conclusion

- ▶ C language and CORBA binding a good choice for DSP applications simply on size alone
  - ▶ The size difference between the same essential set of functionality can be of the order of **5:1**.
    - ▶ e\*ORB C and C++ on Red Hat 9 Linux compiled with gcc 3.2
      - ▶ *C libec\_poa.so 29 kbytes*
      - ▶ *C++ libe\_mpoa.so 105 kbytes*

# CORBA on DSP – Key Challenges

- ▶ Performance
  - ▶ Typically stub and skeleton optimisations only a small part of the overall performance characteristic
  - ▶ Issues such as buffer management, GIOP marshalling and transport efficiency offer more scope for optimisation
  - ▶ No real reason why C++ implementation should not be as performant as C. Performance measurements have shown this to be the case
  - ▶ Transport choice is fundamental in order to meet throughput and latency targets

# CORBA on GPP – Footprint Metrics

## ▶ C++ ORB

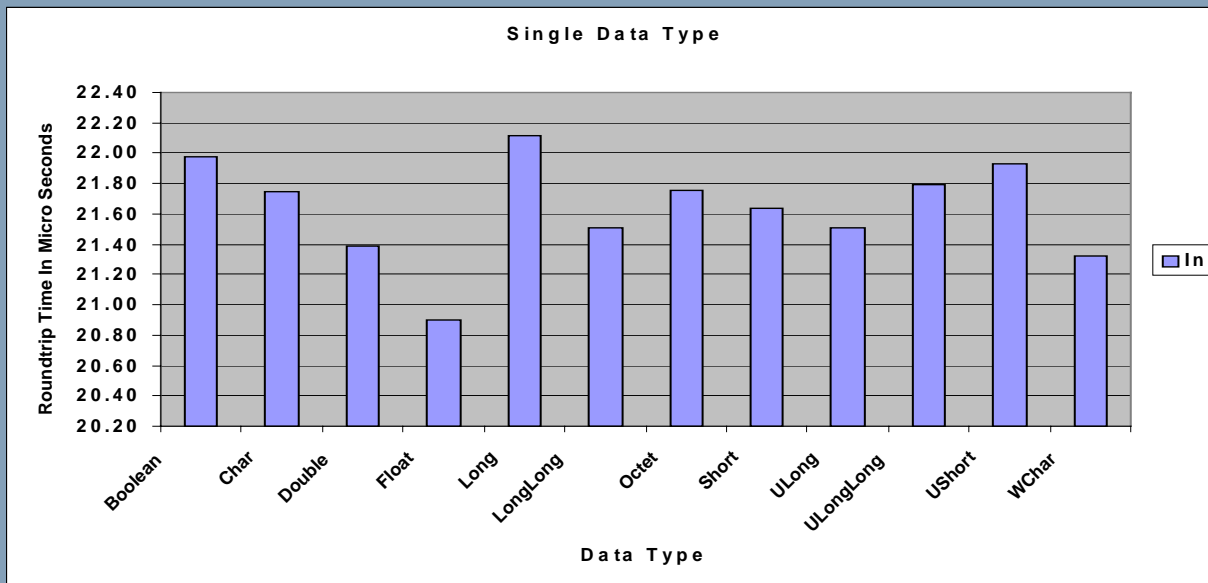
- ▶ **ORB:** OpenFusion e\*ORB SDR v1.0 C++ Edition.
- ▶ **RTOS:** VxWorks 5.4, Tornado 2.0, ccppc 2.7.2
- ▶ **Hardware:** MCPN750 – 1342A 367MHz, 64MB RAM
- ▶ Basic Client and Server Used In ORB Footprint Measurements, void void
  - ▶ Results taken from OpenFusion® e\*ORB SDR v1.0 C++ Edition Footprint Metrics – Version 1.0 Whitepaper

Image	.text	.data	.bss	Static Total (Bytes)	Dynamic Heap (Bytes)	Total ORB Overhead (Bytes)
Kernel	961924	57528	23984	1043436		
Kernel + Server	1229388	65296	25088	1319772		
Kernel + Client	1161436	63128	24816	1249380		
<b>Server Side ORB</b>	267464	7768	1104	276336	49704	<b>326040</b>
<b>Client Side ORB</b>	199512	5600	832	205944	23064	<b>229008</b>

# CORBA on GPP – Performance Metrics

## ▶ C++ ORB

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.0 C++ Edition.
- ▶ **OS:** Red Hat Enterprise Linux Server, WS Release 3 (Taroon Update 3),
- ▶ **Hardware:** Intel Pentium 4 CPU 3.20GHz, 2GBytes RAM, TCP/IP transport.
- ▶ Client and Server roundtrip timings for simple data types

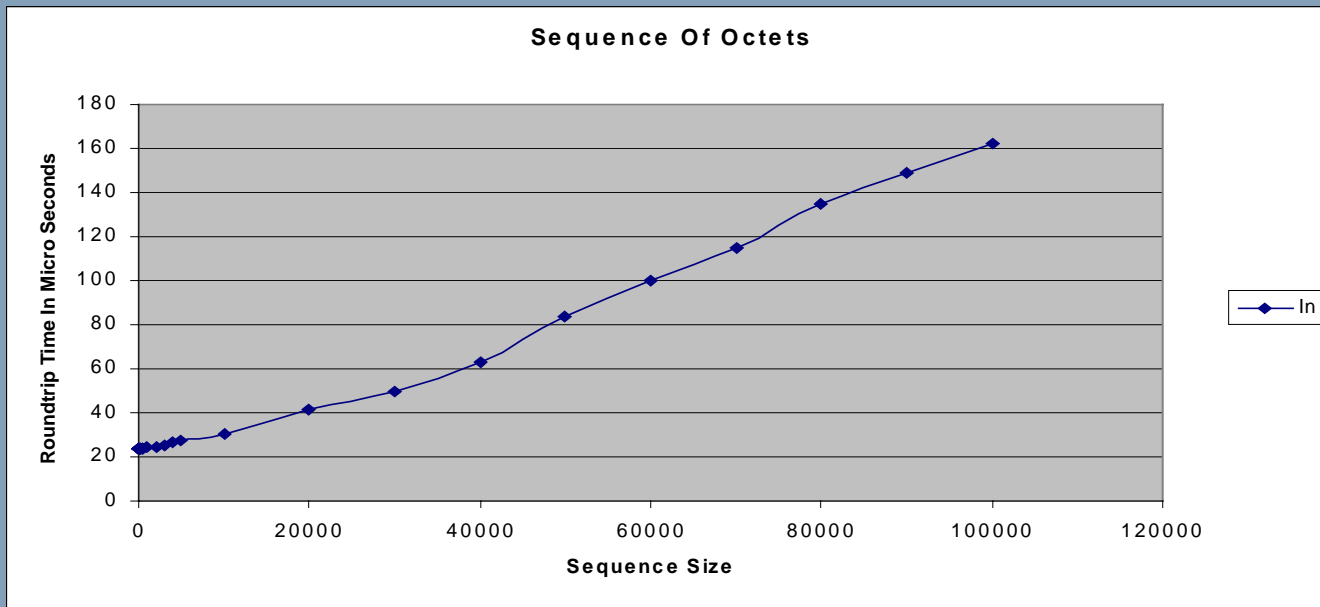


- ▶ **Socket latency = 20.02 micro seconds, therefore ORB overhead represents around only 6% of total round trip time**
  - ▶ Results taken from OpenFusion® e\*ORB SDR v1.0 C++ Edition Footprint Metrics – Version 1.0 Whitepaper

# CORBA on GPP – Performance Metrics

## ▶ C++ ORB

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.0 C++ Edition.
- ▶ **OS:** Red Hat Enterprise Linux Server, WS Release 3 (Taroon Update 3)
- ▶ **Hardware:** Intel Pentium 4 CPU 3.20GHz, 2GBytes RAM, TCP/IP transport.
- ▶ Client and Server roundtrip timings for sequence of octets



- ▶ **Socket latency = 20.02 micro seconds, therefore ORB overhead represents around only 6% overhead of total round trip time**
  - ▶ Results taken from OpenFusion® e\*ORB SDR v1.0 C++ Edition Footprint Metrics – Version 1.0 Whitepaper

# CORBA on GPP – UDP Transport

## ▶ C++ ORB

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.0 C++ Edition.
- ▶ **OS:** Red Hat Enterprise Linux Server, WS Release 3 (Taroon Update 3)
- ▶ **Hardware:** 2 x Intel Pentium 4 CPU 3.20GHz, 2GBytes RAM, 1GB Ethernet, UDP transport
- ▶ Client and Server data throughput – sequence of Octets (1KB data packet) – e\*ORB DIOP implementation requires all data to fit in maximum of 1KB packet

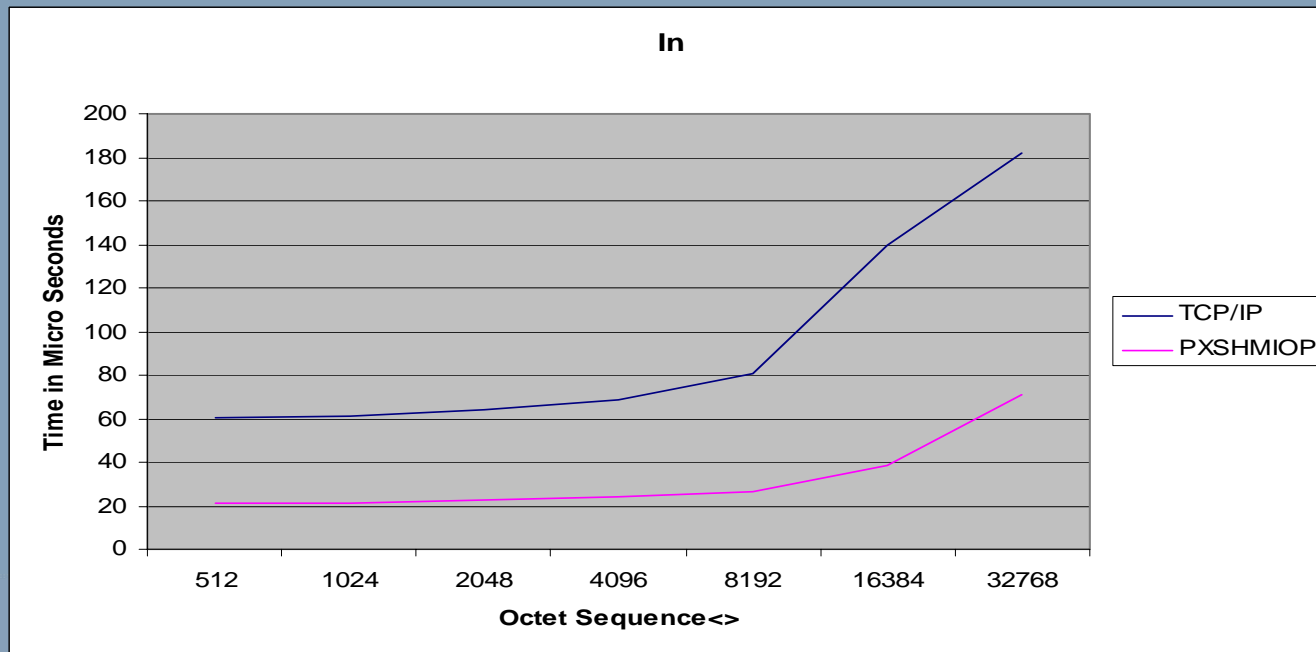
## ▶ Results

- ▶ Data throughput of 51MB/sec, 400Mb/sec possible at 30% packet loss

# CORBA on GPP – Shared Memory Transport

## ▶ C++ ORB

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.0 C++ Edition.
- ▶ **RTOS:** QNX 6.3.0
- ▶ **Hardware:** Intel Pentium 4 CPU 3.20GHz, 2GBytes RAM, POSIX Shared Memory Transport
- ▶ Client and Server roundtrip timings for sequence of octets



# C ORB Footprint Metrics on DSP

## ▶ C ORB

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.0 C Edition
- ▶ **RTOS:** TI BIOS
- ▶ **Hardware:** TI 6416, 1MB on chip memory, TCP/IP Transport
- ▶ Client side ORB footprint measurement – MinimumCORBA “Out-of-the-box” configuration

<b>LIBRARY</b>	<b>CODE</b>	<b>DATA</b>	<b>UDATA</b>	<b>TOTAL</b>
<b>Client ORB Contribution</b>				
ec_iiop.lib	3328	81	48	3457
ec_orb.lib	46848	2643	509	50000
ec_os.lib	3328	162	28	3518
ec_tcp.lib	5248	499	280	6027
				<b>63002</b>
<b>Transport BF3Net</b>				
bf3netlan_wi.l64	134528	948	180	<b>135656</b>
<b>DSPBIOS &amp; Runtime Support</b>				
bios.a64	16160	1450	1120	18730
csl6416.lib	1696	452	344	2492
rts6400.lib	32384	1045	2716	36145
rtdx64xx.lib	3520	84	0	3604
				<b>60971</b>
<b>Total Image Size</b>				<b>259629</b>

# C ORB Footprint Metrics on DSP

## ▶ C ORB

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.0 C Edition
- ▶ **RTOS:** TI BIOS
- ▶ **Hardware:** TI 6416, 1MB on chip memory, TCP/IP Transport
- ▶ Server side ORB footprint measurement – MinimumCORBA “Out-of-the-box” configuration

LIBRARY	CODE	DATA	UDATA	TOTAL
<b>Server ORB Contribution</b>				
ec_iiop.lib	3328	81	48	3457
ec_orb.lib	53664	2805	526	56995
ec_os.lib	3328	162	28	3518
ec_poa.lib	16128	1016	292	17436
ec_tcp.lib	5248	499	280	6027
				<b>87433</b>
<b>Transport BF3Net</b>				
bf3netlan_wi.l64	134528	948	180	<b>135656</b>
<b>DSPBIOS &amp; Runtime Support</b>				
bios.a64	16160	1450	1120	18730
csl6416.lib	1696	452	344	2492
rtdx64xx.lib	3520	84	0	3604
rts6400.lib	32704	1061	2724	36489
				<b>61315</b>
<b>Total Image Size</b>				<b>284404</b>

# C++ ORB Footprint Metrics on DSP

## ▶ C++ ORB

- ▶ ORB: OpenFusion e\*ORB SDR v1.0 C++ Edition
- ▶ RTOS: TI BIOS
- ▶ Hardware: TI 6416, 1MB on chip memory, TCP/IP Transport
- ▶ Client side ORB footprint measurement – MinimumCORBA “Out-of-the-box” configuration

LIBRARY	CODE	DATA	UDATA	OTHER	TOTAL
<b>Client ORB Contribution</b>					
e_giop.lib	81792	2158	200	5908	90058
e_iiop.lib	704	101	56		861
e_orb.lib	100352	4412	852	8589	114205
e_reactor.lib	4160	100	8	492	4760
ec_os.lib	2720	142	14		2876
e_tcp.lib	37120	1235	168	2189	40712
					<b>253472</b>
<b>Transport NDK</b>					
hal_eth_mxf.lib	8698	56	384	1536	10584
hal_ser_stub.lib	64	0	0		64
hal_timer.lib	416	12	28		456
hal_userled.lib	32	12	4		48
netctrl.lib	4832	267	372		5471
nettool.lib	31808	971	845		33624
os_sem.lib	13856	878	420	136668	151822
stk.lib	82848	1854	1429		86131
					<b>288200</b>
<b>DSPBIOS &amp; Runtime Support</b>					
bios.a64	22016	1534	1120		24670
csl6416.lib	6368	896	692		7956
lnkrtdx.a64	1344	92	24		1460
rts6400.lib	39136	1093	2733		42962
rtdx64xx.lib	3520	84	0		3604
					<b>80652</b>
<b>Total Image Size</b>					<b>622324</b>

# C++ ORB Footprint Metrics on DSP

## ▶ C++ ORB

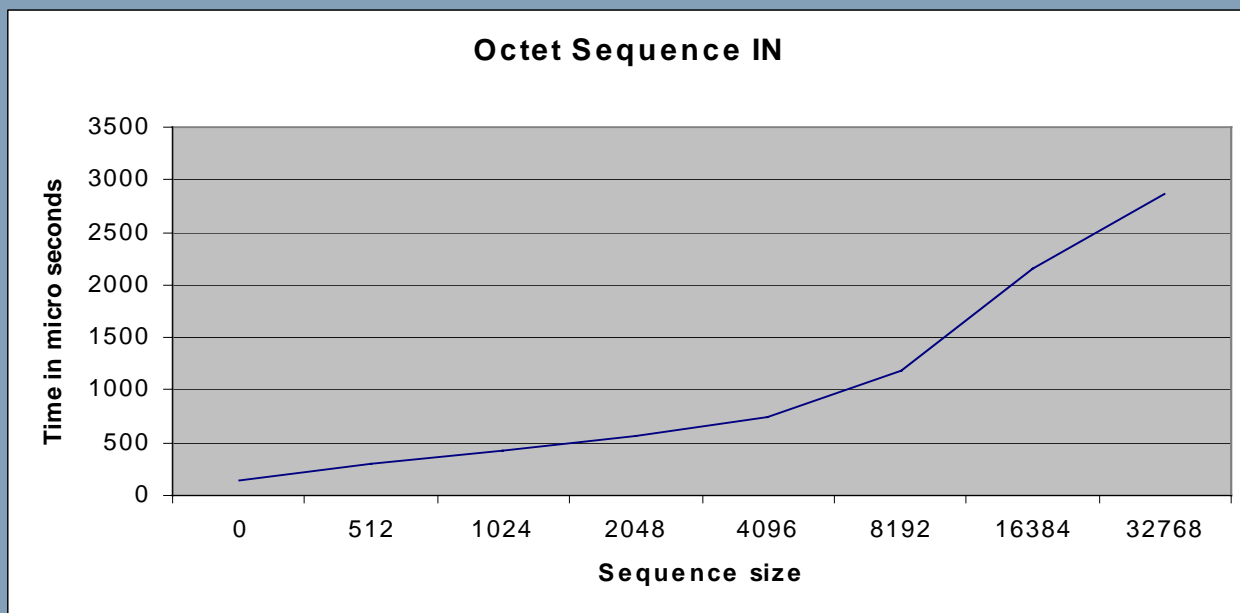
- ▶ ORB: OpenFusion e\*ORB SDR v1.0 C++ Edition
- ▶ RTOS: TI BIOS
- ▶ Hardware: TI 6416, 1MB on chip memory, TCP/IP Transport
- ▶ Server side ORB footprint measurement – Minimum CORBA “Out-of-the-box” configuration

LIBRARY	CODE	DATA	UDATA	OTHER	TOTAL
<b>Server ORB Contribution</b>					
e_giop.lib	81792	2158	200	5908	90058
e_iiop.lib	704	101	56		861
e_m_poa.lib	83456	2351	356	24206	110369
e_orb.lib	104064	4640	876	8589	118169
e_reactor.lib	4160	100	8	492	4760
ec_os.lib	2976	142	14		3132
e_tcp.lib	37120	1235	168	2189	40712
					<b>368061</b>
<b>Transport NDK</b>					
hal_eth_mxf.lib	8698	56	384	1536	10584
hal_ser_stub.lib	64	0	0		64
hal_timer.lib	416	12	28		456
hal_userled.lib	32	12	4		48
Netctrl.lib	4832	267	372		5471
nettool.lib	31808	971	845		33624
os_sem.lib	13856	878	420	136668	151822
stk.lib	82848	1854	1429		86131
					<b>288200</b>
<b>DSPBIOS &amp; Runtime Support</b>					
bios.a64	22016	1534	1120		24670
csl6416.lib	6368	896	692		7956
lnkrtdx.a64	1344	92	24		1460
rts6400.lib	39136	1093	2733		42962
rtdx64xx.lib	3520	84	0		3604
					<b>80652</b>
<b>Total Image Size</b>					<b>736913</b>

# CORBA on DSP – Performance Metrics

## ▶ C ORB

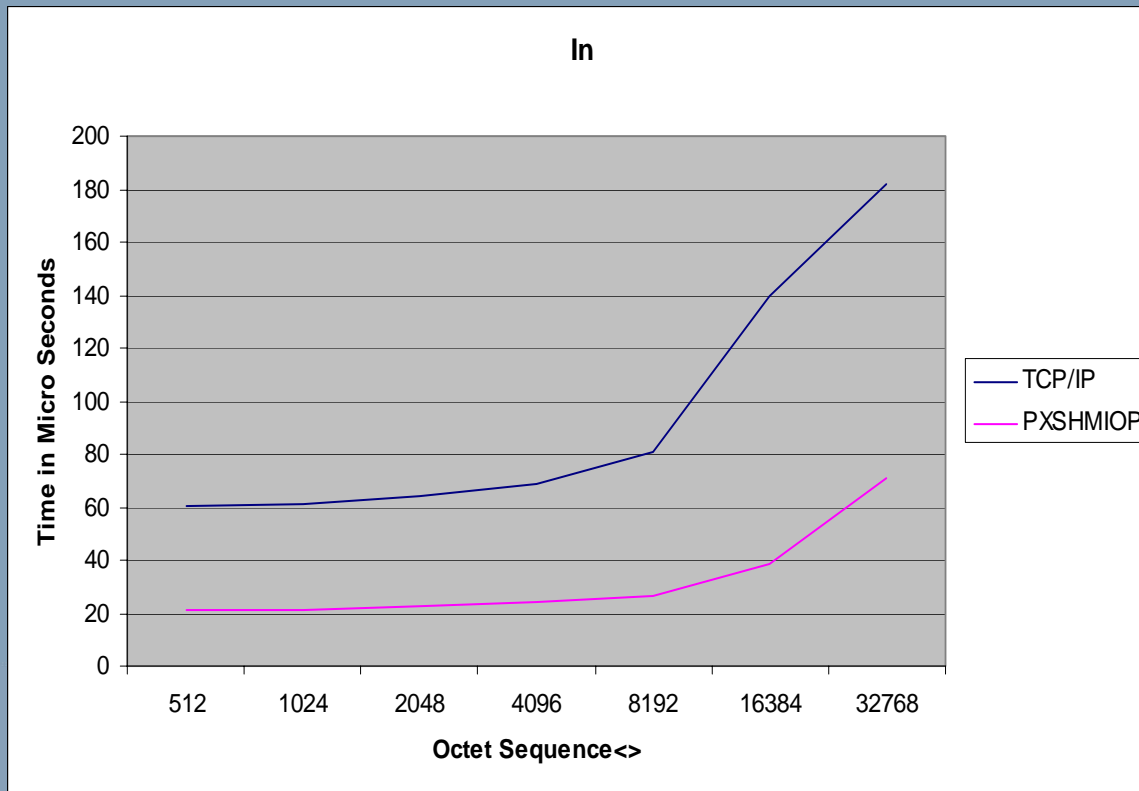
- ▶ **ORB:** OpenFusion e\*ORB SDR v1.0 C Edition.
- ▶ **OS:** Client side: Red Hat Enterprise Linux Server, WS Release 3 (Taroon Update 3), Server Side: TI BIOS
- ▶ **Hardware:** Client Side: Intel Pentium 4 CPU 3.20GHz, 2GBytes RAM, TCP/IP transport, Server Side: TI 64161MB on chip memory, NDK TCP/IP Transport
- ▶ Client and Server roundtrip timings for sequence of octets



- ▶ **Socket latency = 138 micro seconds, therefore ORB overhead represents around only 8% overhead of total round trip time**

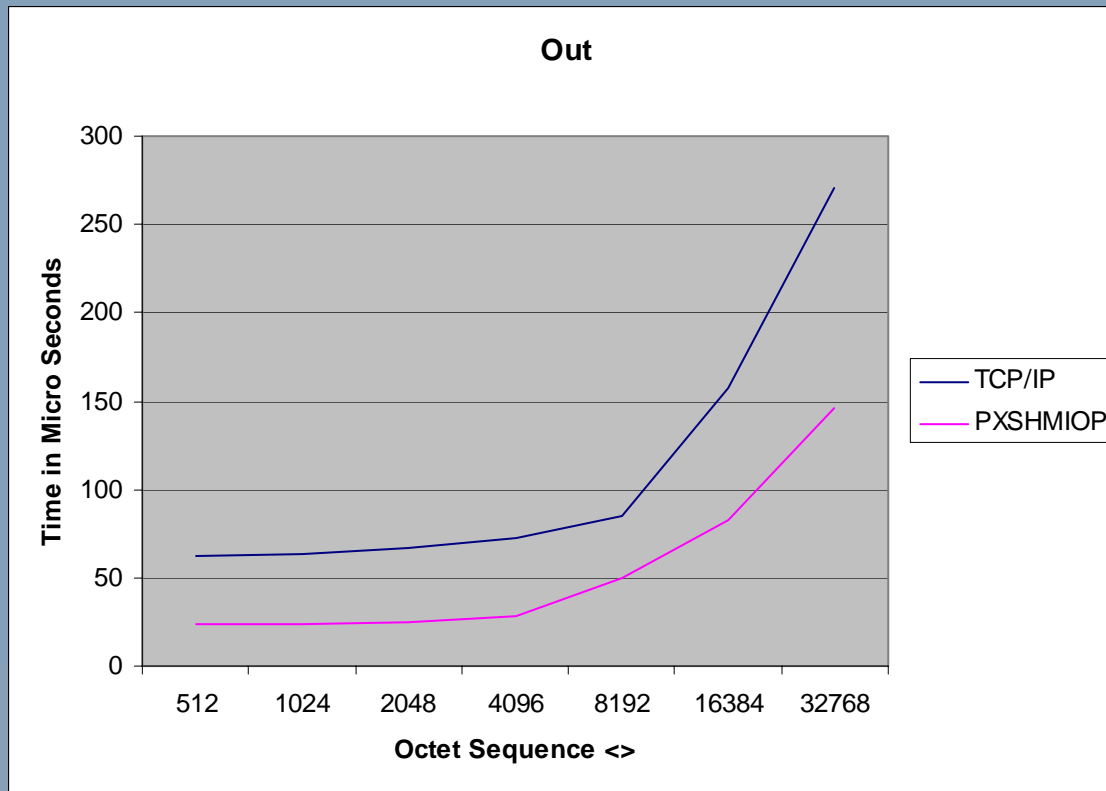
# Shared Memory Performance Metrics (In Parameter)

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.1 C++ Edition.
- ▶ **RTOS:** QNX 6.3.0
- ▶ **Hardware:** x86, Pentium P4, 3.2 Ghz



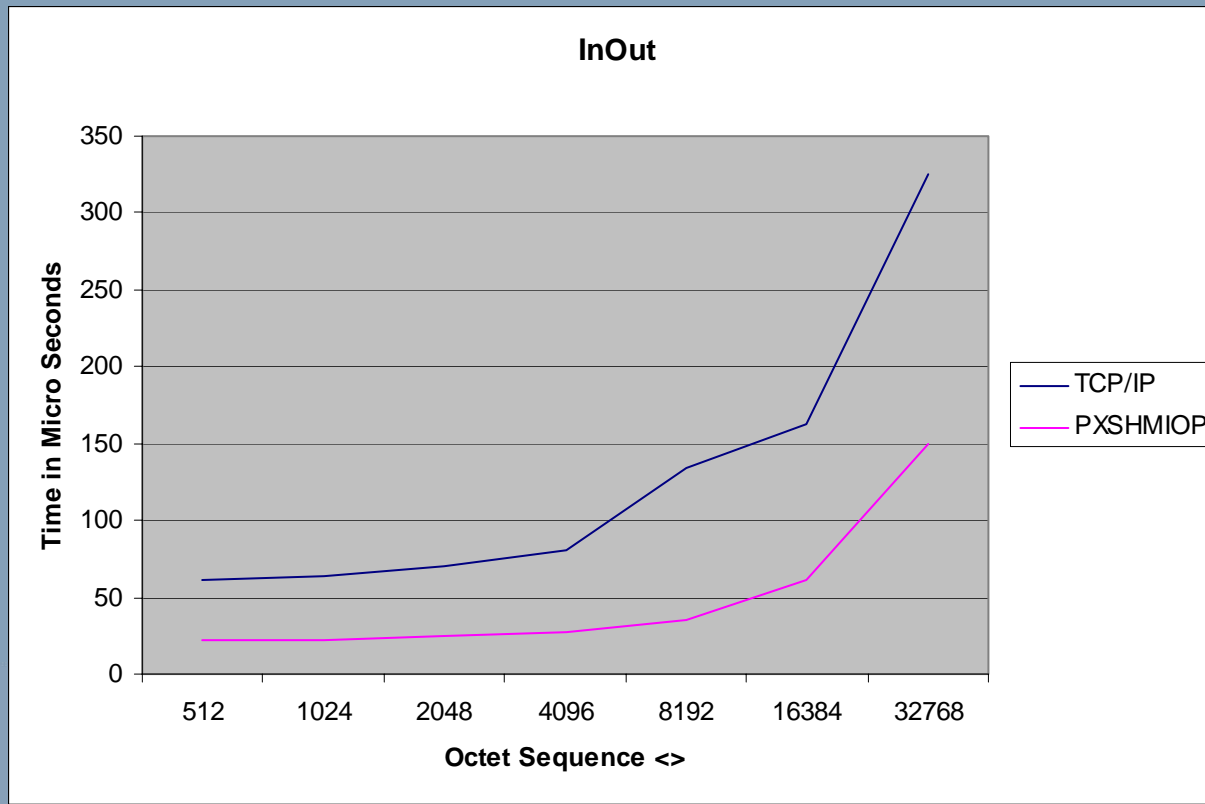
# Shared Memory Performance Metrics (Out Parameter)

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.1 C++ Edition.
- ▶ **RTOS:** QNX 6.3.0
- ▶ **Hardware:** x86, Pentium P4, 3.2 Ghz



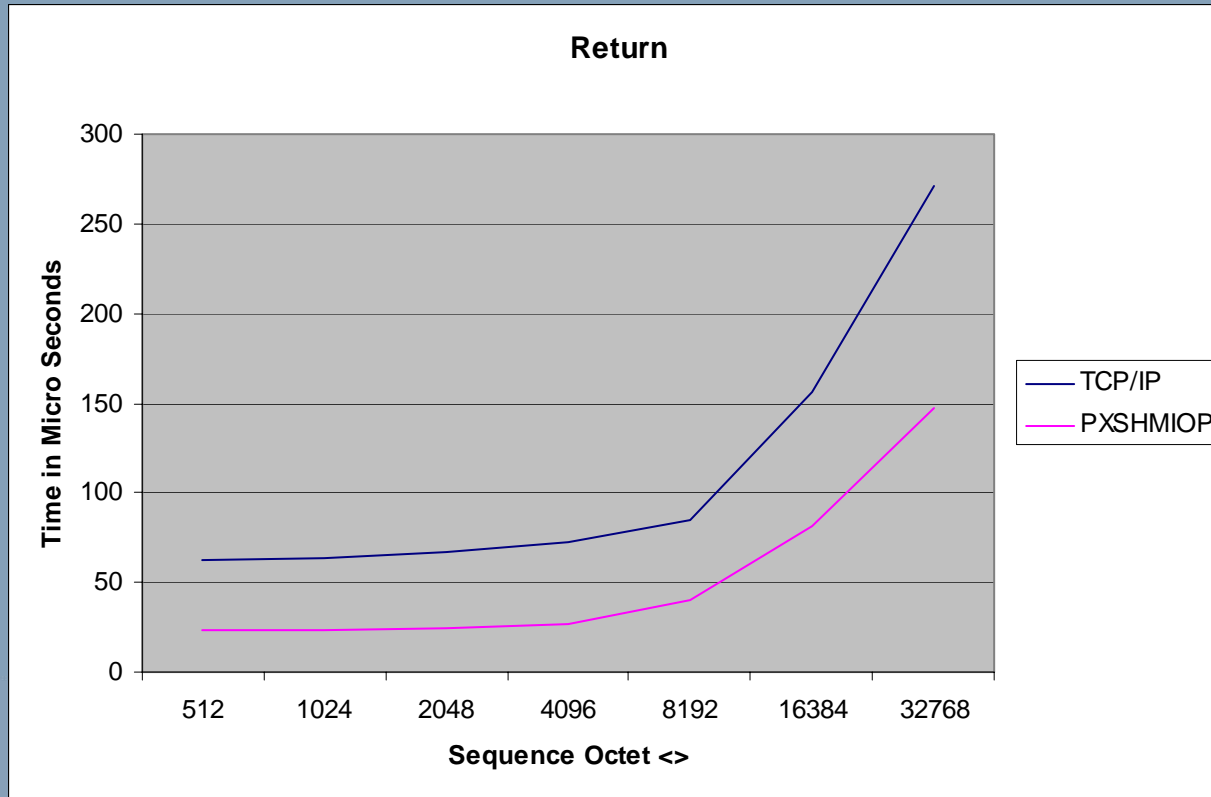
# Shared Memory Performance Metrics (Inout Parameter)

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.1 C++ Edition.
- ▶ **RTOS:** QNX 6.3.0
- ▶ **Hardware:** x86, Pentium P4, 3.2 Ghz



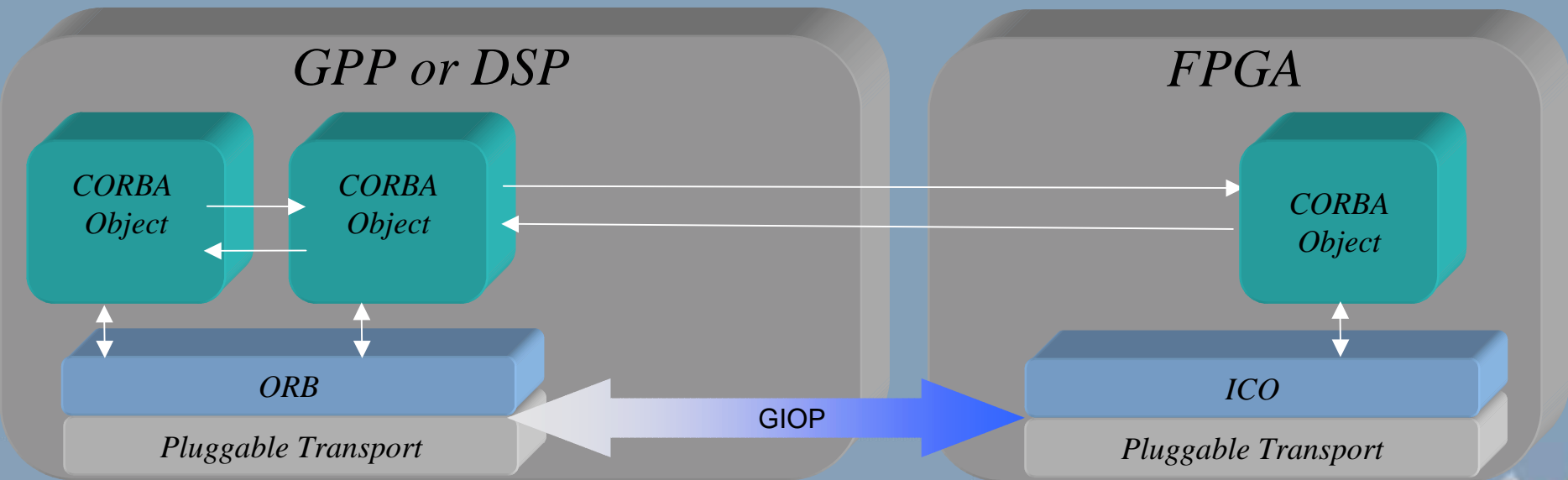
# Shared Memory Performance Metrics (Return)

- ▶ **ORB:** OpenFusion e\*ORB SDR v1.1 C++ Edition.
- ▶ **RTOS:** QNX 6.3.0
- ▶ **Hardware:** x86, Pentium P4, 3.2 Ghz



# Integrated Circuit ORB

- ▶ SCA compliance is maintained to the FPGA
- ▶ Transport overhead is reduced
  - ▶ 100x faster than S/W proxy implementation



# Why use an ICO?

- ▶ Software developers generally limited CORBA's use, in most cases, to implementations executing on General Purpose Processors (GPPs).
- ▶ CORBA benefits becoming important as system complexity drives the use of DSPs and FPGAs
  - ▶ Location Transparency, Server Transparency, Language Independence, Implementation Independence, Architecture Independence, Protocol Independence and Transport Independence
- ▶ Pros and Cons of running CORBA in a FPGA embedded processor versus direct use of VHDL language bindings.
  - ▶ as portability, reuse, and elimination of software proxies).

# Why use an ICO?

- ▶ Use of an embedded processor in an FPGA for the sole purpose of supporting middleware is impractical
  - ▶ FPGAs that support embedded processors are at the high end of the price range
  - ▶ Running an ORB in an FPGA embedded processor uses significant memory resources
    - ▶ Wasting valuable FPGA resources on CORBA functionality
  - ▶ Choice of industry standard embedded processors is limited
    - ▶ Most are custom processors locked to particular FPGA vendor
  - ▶ Embedded processor performance is typically poor
    - ▶ Most embedded processors are clocked at significantly lower speeds than commercially available GPPs and DSPs

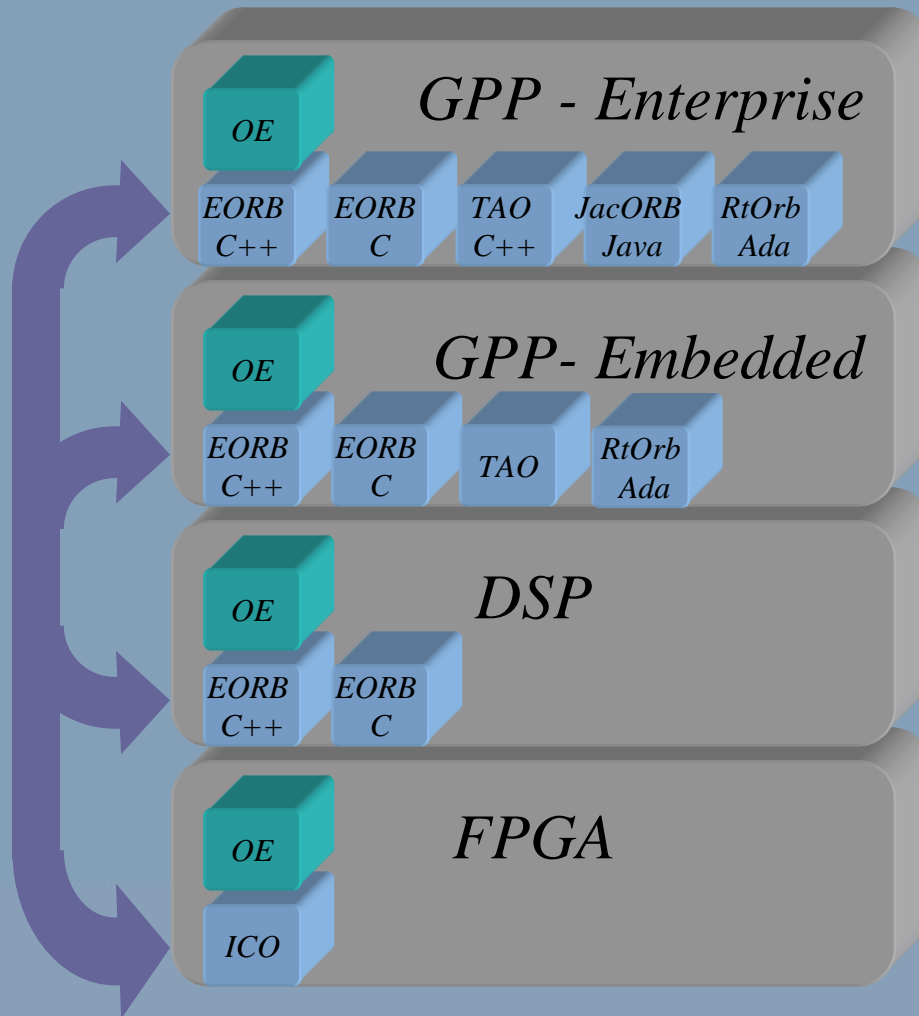
# Why use ICO?

- ▶ ICO uses no block memory and currently uses fewer than 3000 logic cells and is portable between FPGA families
- ▶ ICO improves portability and re-use in SCA-based applications since the coupling between software on GPP/DSPs and FPGAs is eliminated
  - ▶ Coupling via Memory Mapped I/O, for example
- ▶ Eliminates the need for proxies/adapters
  - ▶ Reduces overhead, latency
  - ▶ Increases throughput
- ▶ Eliminates the need for complex hardware abstraction layer protocols (Supports direct access to SCA components running on H/W)
  - ▶ Spectra tools auto-generate VHDL delivering H/W SCA “components”
- ▶ Applications in security-related areas where the assurance of large software applications (such as ORBs) is suspect

- ▶ The latency through the ICO from the time a GIOP message arrives at its inputs until the response message is ready at its output is about .200ns.



# SCA Operating Environment (OE)



# SCA Operating Environment (OE)

- ▶ Implementation of SCA 2.2 specification
  - ▶ CORBA components using e\*ORB C Edition
    - ▶ Common platforms: Linux, Integrity, VxWorks, TI DSP/BIOS, OSE
  - ▶ Small footprint
    - ▶ Approximately 1Mb
  - ▶ Portable
    - ▶ Abstraction layers for system specific APIs (non POSIX)
- ▶ Core framework elements written in C
  - ▶ CF::DomainManager
  - ▶ CF::DeviceManager
  - ▶ CF::ExecutableDevice
  - ▶ CF::FileSystem
  - ▶ CF::File
  - ▶ CF::ApplicationFactory

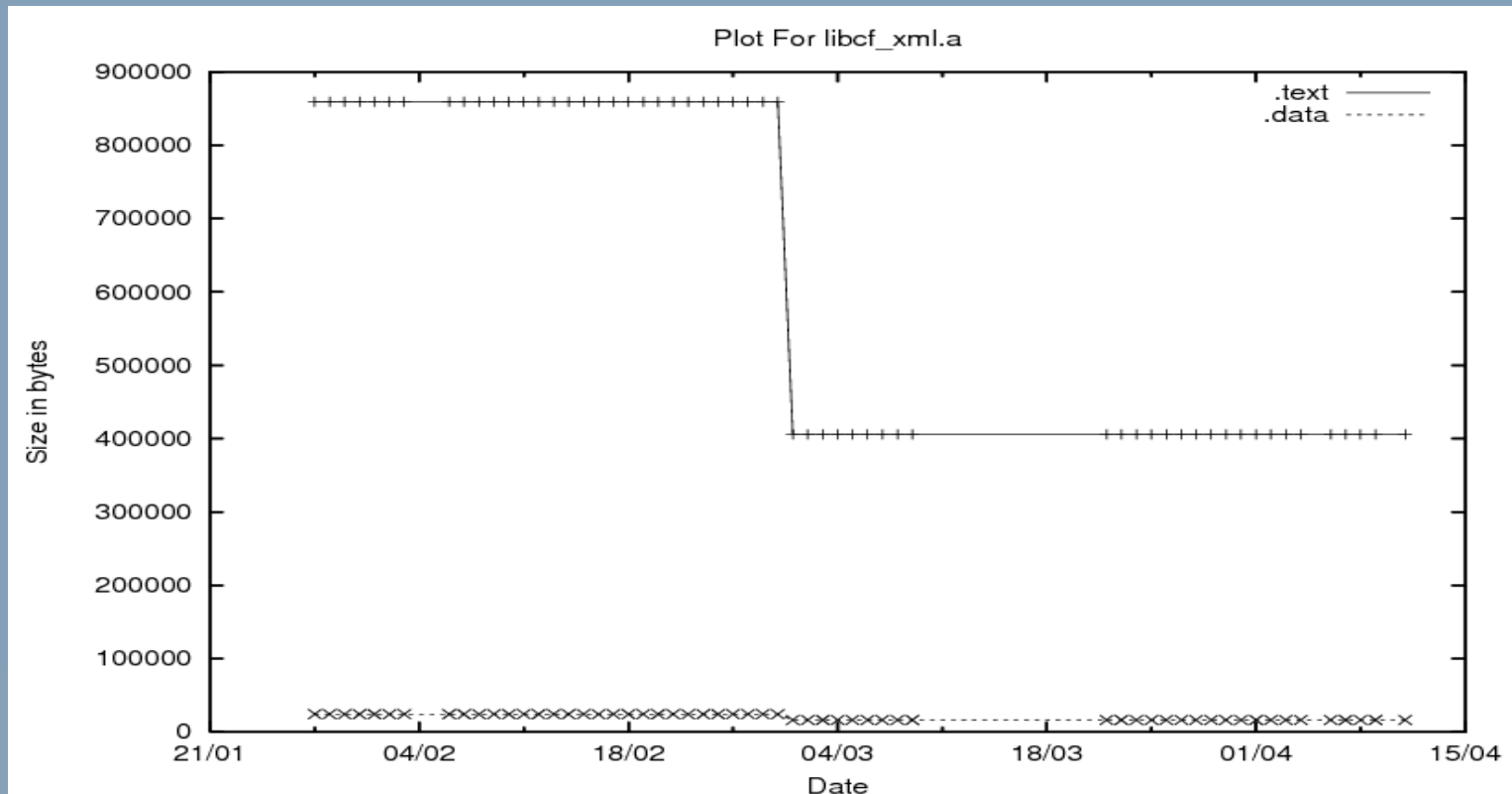
# SCA Operating Environment (OE)

- ▶ Spectra Tools generates
  - ▶ CF::Resource
  - ▶ CF::ResourceFactory
  - ▶ XML domain profiles
- ▶ Spectra Tools generators
  - ▶ C++ component generation supported
  - ▶ C component generation in development
- ▶ Lightweight Services
  - ▶ Name
  - ▶ Log
  - ▶ Event



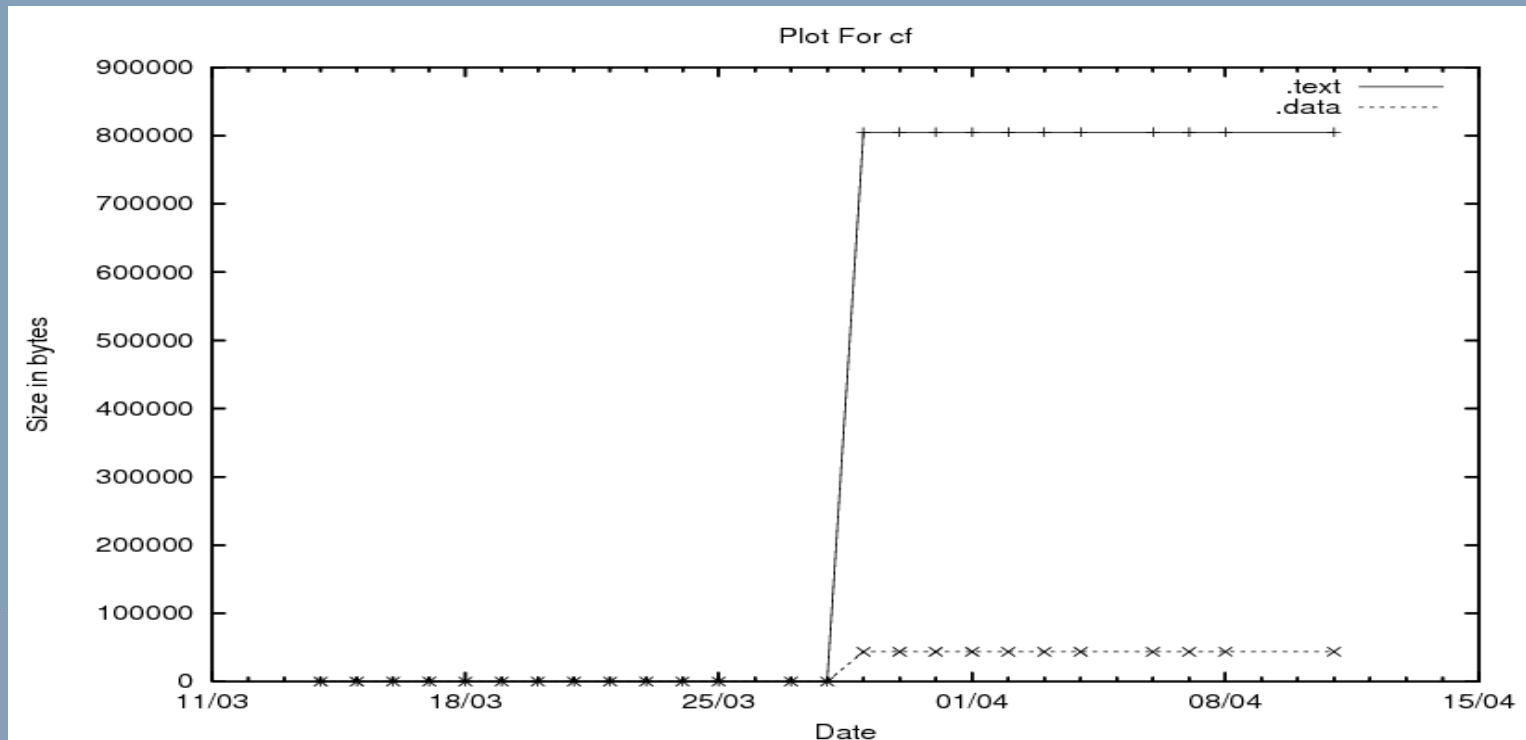
# XML Parser - Size

- ▶ XML Parser contributing ~ 400K to overall image size
  - ▶ More future optimisation of XML library planned



# Core Framework – Size

- ▶ Total Library sizes for Core Framework
  - ▶ libcf\_impl.so
  - ▶ libcf\_logservice\_c.so
  - ▶ libcf\_logservice\_s.so
  - ▶ libcf\_xml.so



# Benefits of using CORBA in SDR

- ▶ CORBA is small and fast enough to efficiently support communication across the whole signal processing chain, including FPGA and DSP environments
- ▶ Efficient CORBA implementations like e\*ORB impose little overhead on top of the underlying performance of the transport
- ▶ Choice of transports in ORB is critical to meeting performance criteria
  - ▶ ETF allows for custom transports to be easily supported
  - ▶ ETF allows for multiple transports to be configured in and used in the same system
- ▶ The language neutrality of CORBA allows OE to be written in C (very low footprint) but still support waveforms written in other languages such as C++ and ADA

# Benefits of using CORBA in SDR

- ▶ If CORBA is not used ...
  - ▶ You're on the road to a poor man's CORBA
    - ▶ Still have to solve the same issues in a proprietary way
      - ▶ *Transports*
      - ▶ *Message formats*
      - ▶ *Marshalling/Unmarshalling of types*
      - ▶ *Call dispatch*
- ▶ The benefits of CORBA are substantial
  - ▶ Facilitates implementation of portable waveforms. A key goal of the JTRS program
  - ▶ The use of standards based middleware like CORBA and SCA enables greater tool integration, supporting faster development through MDD and generative programming techniques