Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Studio e analisi di software per il porting di codice a 64 bit

Relatore: Chiar.mo
Prof. GIULIO CASCIOLA

Presentata da:
ANDREA CHIACCHIERA

III Sessione
Anno Accademico 2013-14

Alla mia famiglia...

Introduzione

XCModel è un sistema CAD, basato su NURBS, realizzato ed utilizzato in ambiente accademico.
È composto da quattro pacchetti per la modellazione 2D, 3D e la resa fotorealistica, ognuno dotato di una propria interfaccia grafica.
Questi pacchetti sono in costante evoluzione: sia per le continua evoluzioni dell'hardware che ai cambiamenti degli standard software. Il sistema nel complesso raccoglie la conoscenza e l'esperienza nella modellazione geometrica acquisita nel tempo dai progettisti[CASC01].
XCModel, insieme ai suoi sottosistemi, sono stati progettati per diventare un laboratorio di insegnamento e ricerca utile a sperimentare ed imparare metodi ed algoritmi nella modellazione geometrica e nella visualizzazione grafica.[CASC01]
La natura principalmente accademica, e la conseguente funzione divulgativa, hanno richiesto continui aggiornamenti del programma affinché potesse continuare a svolgere la propria funzione nel corso degli anni. La necessità di continuare a ad evolversi, come software didattico, anche con il moderno hardware, è forse il principale motivo della scelta di convertire XCModel a 64 bit; una conversione che ho svolto in questa tesi.
Come molte altre applicazioni realizzate a 32 bit, la maggior parte del codice viene eseguito correttamente senza problemi. Vi sono però una serie di problematiche, a volte molto subdole, che emergono durante la migrazione delle applicazioni in generale e di XCModel in particolare.
Questa tesi illustra i principali problemi di portabilità riscontrati durante il

porting a 64 bit di questo pacchetto seguendo il percorso da me intrapreso: mostrerò gli approcci adottati, i tool utilizzati e gli errori riscontrati.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

XCModel

Il progetto XCModel è un sistema basato su NURBS creato per modellare curve e superfici, comporre oggetti solidi e rendere le scene modellate.
Nato oltre dieci anni fa, si è perfezionato nel tempo grazie al contributo di varie persone.
XCModel è scritto in C ed è progettato per funzionare su sistemi operativi Unix dotati del sistema Xwindow; è anche in grado di operare su varie piattaforme:

- SUN Sparc - Solaris
- SGI - Irix
- Intel - Linux

A differenza degli altri sistemi CAD, XCModel non utilizza l'accelerazione grafica hardware ma il solo processore per compiere tutti i calcoli necessari alla modellazione ed alla resa. Si noti bene: questa non è una carenza ma una precisa scelta progettuale che rende XCModel un sistema didattico a livello universitario.
Il codice è stato creato non tanto per essere il più efficiente possibile, ma piuttosto per essere un ottimo *case study* per gli studenti, che possono così apprendere gli algoritmi che stanno dietro alla computer grafica, modificarli

e provarne di nuovi.
Il codice sorgente infatti può contenere diverse versioni dello stesso algoritmo rendendo il sistema XCModel più simile ad una libreria piuttosto che ad un software professionale.
Nonostante questo le prestazioni non sono affatto minori ad un CAD professionale, soprattutto nella modellazione dove la risposta del sistema è praticamente istantanea.
XCModel, definito come *aCADemic modelling/rendering system*, è principalmente rivolto agli studenti ma può essere utilizzato nella realizzazione di modelli professionali.
Il pacchetto è composto da 162477 linee di codice per un totale di 4768618 byte (più di 4,5 Mb), una volta installato e compilato, nella versione a 32 bit, XCModel occupa circa 12Mb. Un progetto che ha richiesto la ricerca di tecniche di migrazione adeguate alla complessità dello stesso: per sua natura XCModel non ha specifiche di realizzazione, ma solo linee guida fornite dal *project designer* (ruolo attualmente ricoperto dal prof. Giulio Casciola), una assenza che rende il sorgente di questo pacchetto una eterogeneità di stili di programmazione. Ogni studente che apporta il proprio contributo al progetto fornisce anche il proprio stile di programmazione.

L'ambiente XCModel è composto da quattro pacchetti dotati di una sofisticata interfaccia grafica interattiva:

- XCCurv - esegue la modellazione di curve NURBS bidimensionali,
- XCSurf - esegue la modellazione di curve NURBS e superfici tridimensionali,
- XCBool - effettua operazioni booleane sulle superfici (unione, intersezione, differenza, ecc.)
- XCRayt - esegue la descrizione e il rendering di una scena modellata da un algoritmo di ray-tracing e ne effettuano la visualizzazione.

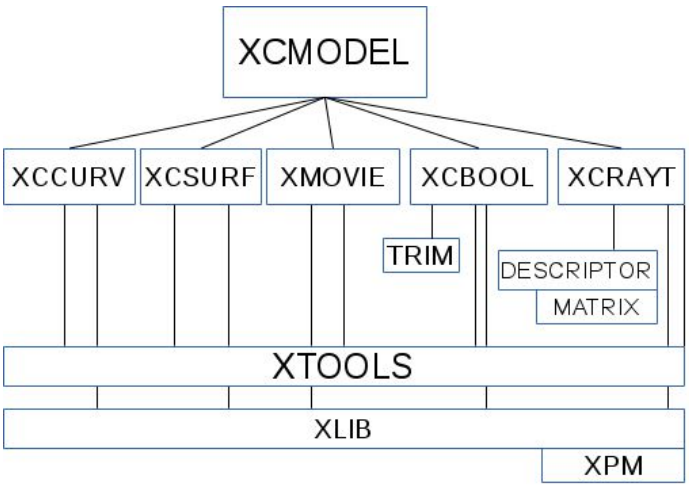

Figura 1.1: Struttura del programma XCModel

- XCView - visualizzazione di immagini

Tutti questi pacchetti utilizzano le funzioni della Libreria XTools, la quale fornisce numerose primitive grafiche facenti uso delle potenzialità di Xwindow.

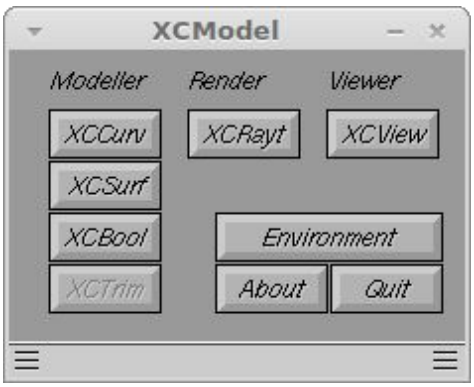

Figura 1.2: Finestra di avvio di XCModel

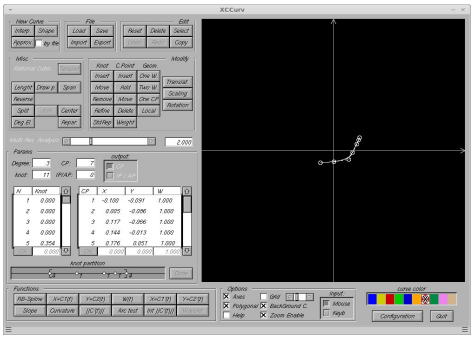

Figura 1.3: Finestra di avvio di XCCurv

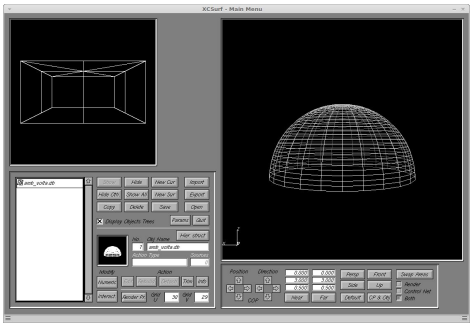

Figura 1.4: Finestra di avvio di XCSurf

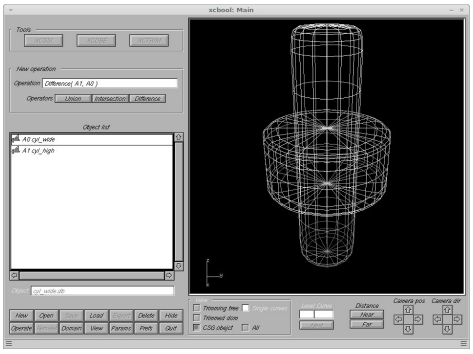

Figura 1.5: Finestra di avvio di XCBool

1.1 XTools

XTools è una libreria creata per sistemi Unix con ambiente Xwindow. È stata creata per produrre una interfaccia grafica più semplice di quelle create con Xwindow. Questa libreria dispone di un numero limitato di oggetti che possono essere suddivisi in due classi: primitive e derivate. Le prime sono i costituenti base di una interfaccia grafica, come pulsanti, etichette, caselle di testo, finestre etc. Le seconde sono costruite dalle prime e forniscono specifiche funzioni: *help text* (informazioni sulla barra di stato), *file request* (finestra di dialogo per inserire un file) e la *message box* (box informativo).

1.2 Esempi di modelli

Con XCModel si riescono a progettare anche modelli complessi, come quelli realizzati nell'ambito del corso di Grafica. Nella figura 1.6 si può vedere un modello di Ferrari composto da 105 elementi tutti prodotti a partire da curve 2D con i pacchetti xccurv e xcsurf. La resa è stata prodotta con il pacchetto xcrayt e una immagine è mostrata in figura 1.7.

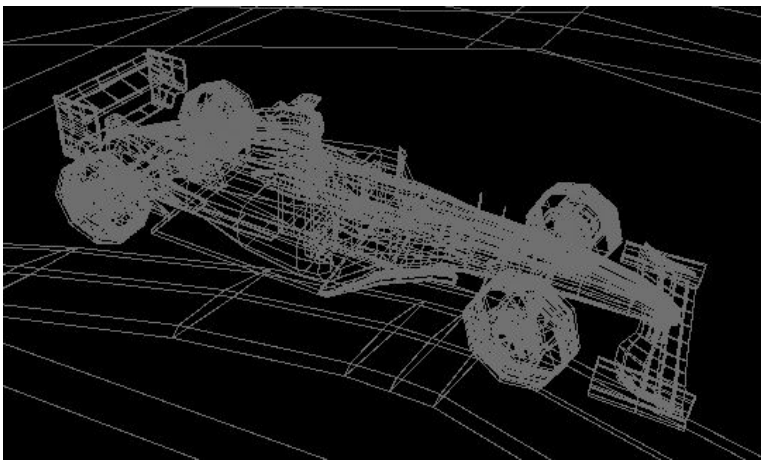

Figura 1.6: Modellazione

Figura 1.7: Resa

Capitolo 2

Architettura a 64 bit

L'architettura a 64 bit è resa necessaria dalle richieste delle applicazioni che necessitano di un maggiore spazio di indirizzi: principalmente i server ad alte prestazioni, i data manager, i CAD e, naturalmente, i giochi. Sono queste le applicazioni che maggiormente beneficeranno dello spazio di indirizzamento a 64 bit e del maggior numero di registri, la cui ridotta quantità, nell'architettura x86, ne limitava lo svolgimento delle attività di elaborazione.
Esistono diverse architetture a 64 bit, ma le due più popolari sono: IA64 e Intel 64.

IA-64 : è una architettura a 64 bit sviluppato congiuntamente da Intel e Hewlett Packard, implementata nei processori della serie Itanium. Inizialmente gli unici processori 64 bit prodotti da Intel.

x64 : è un'estensione dell'architettura x86 con piena retro-compatibilità al set di istruzioni x86, sviluppata inizialmente da AMD ed integrata successivamente da Intel nei propri IA32 chiamandoli così Intel64.

Queste due architetture assolutamente diverse ed incompatibili tra loro.

2.1 Tipi di dati

La tabella 2.1 mostra i diversi modelli di dati nelle architetture a 32 e 64 bit.

Modello dei dati	ILP32	LP64 o I32LP64	LLP64 o IL32P64	ILP64	SILP64
char	8	8	8	8	8
short	16	16	16	16	16
int	32	32	32	64	64
long	32	64	32	64	64
long long	64	64	64	64	64
size_t	32	64	64	64	64
puntatore	32	64	64	64	64
	32 bit UNIX	Unix, Unix like, OS X	Ms Windows	porting di Solaris su SPARC64	UNICOS

Tabella 2.1: Tipi di dati nelle diverse architetture

2.2 AMD64

L'architettura x86-64 (nota anche come x64, Intel64, EMT64T e AMD64) è la versione a 64 bit del set di istruzioni x86. Riesce a supportare una maggiore quantità di memoria virtuale e fisica (in teoria, 2^{64} byte ovvero 16 exbibyte) rispetto ai suoi predecessori a 32 bit, possiede registri *general pourpose* a 64 bit ed offre numerosi altri miglioramenti. La specifica originale, creata da AMD, è stata implementata da AMD, Intel e VIA . È pienamente retro-compatibile con il codice x86 a 16 e 32 bit, e poiché i set di istruzioni x86 sono implementati in hardware, senza alcuna emulazione, le istruzioni

x86 vengono eseguite senza limitazioni di compatibilità o di prestazioni.
I processori Amd64 ed Intel64 possono operare in due modalità:

modalità long - una modalità a 64 bit dove possono essere eseguite le applicazioni a 16 e 32 bit;

modalità legacy - in cui il processore viene eseguito come se fosse un vecchio processore a 32 bit dove non è possibile accedere alle funzionalità a 64 bit.

Una caratteristica dell'architettura Amd64 è la presenza di 16 registri *general purpose* a 64 bit (erano otto a 32-bit nell'architettura x86-32). È presente anche il supporto a 64 bit per operazioni aritmetiche e logiche su numeri interi.
I principali vantaggi che possiamo trovare nell'architettura x86-64 sono:

- uno spazio di indirizzamento a 64 bit,
- un set registro esteso,
- un set di istruzioni familiare agli sviluppatori,
- la capacità di applicazioni eseguire a 16 e 32 bit in un sistema operativo a 64 bit,
- la capacità di utilizzare sistemi operativi a 32 bit.

2.3 Sistemi operativi a 64-bit

Quasi tutti i moderni sistemi operativi presentano una versione per l'architettura x86-64. Ciò non significa che l'intero codice di un tale sistema sia a 64 bit. Alcune parti del sistema operativo e molte applicazioni potrebbero rimanere a 32 bit.

2.3.1 Win64

Come in Win32, la dimensione di una pagina di memoria in Win64 è 4 Kbyte. I primi 64 Kbyte di spazio di indirizzamento non vengono mai visualizzati: il più basso indirizzo utilizzabile è *0x10000*. I compilatori per Intel64 hanno una particolarità: possono utilizzare i registri con grande efficienza per passare parametri alle funzioni invece di utilizzare lo stack. Questo ha permesso agli sviluppatori dell'architettura Win64 di eliminare il concetto di convenzione di chiamata: mentre in Win32 è possibile utilizzare diverse convenzioni, in Win64 ne è presente solo una.
La differenza di convenzioni di chiamata rende impossibile l'utilizzo di codice sia a 64 bit che a 32 bit in un unico programma: se un programma è stato compilato per la modalità a 64 bit, anche le librerie utilizzate devono essere a 64 bit. Il passaggio di parametri tramite i registri è una delle innovazioni che rendono i programmi a 64 bit più veloci di quelli a 32 bit. Si può ottenere un ulteriore guadagno prestazioni utilizzando i tipi di dati a 64 bit.

Area di indirizzi

Anche se un processore a 64 bit può indirizzare teoricamente 16 Ebytes di memoria (2^{64}), Win64 ora supporta solo 16 Tbyte (2^{44}), limitata anche dagli attuali processori in grado di fornire l'accesso solo a un Tbyte (2^{40}) di memoria fisica. L'architettura del sistema operativo (non l'hardware) può estendere questo spazio fino a 4 Pbyte (2^{52}), sebbene in questo caso si renda necessaria una grande quantità di memoria per le tabelle di pagina che lo rappresentano.

2.3.2 Linux

Linux è diventato un sistema operativo a 64 bit da quando è stata portato sulla piattaforma HP Alpha nel 1993. Allora la piattaforma di riferimento di Linux era a 32 bit su processore IA32. Come UNIX ed altri sistemi operativi

UNIX-like, Linux utilizza lo standard LP64 dove puntatori e interi lunghi sono a 64 bit, mentre gli interi (*int*) rimangono a 32-bit.

Capitolo 3

Applicazioni a 32 bit in ambiente a 64 bit

3.1 Windows

La compatibilità delle applicazioni a 32 bit sulle versioni di Windows a 64 bit è resa possibile attraverso il sistema *WoW64*. WoW64 (Windows-on-Windows a 64 bit) è un sottosistema del sistema operativo Windows che consente di eseguire applicazioni a 32 bit su tutte le versioni a 64 bit di Windows. Il sottosistema WoW64 non supporta i seguenti programmi:

- programmi compilati per i sistemi operativi a 16-bit,
- programmi in modalità kernel compilati per i sistemi operativi a 32 bit.

Diversi processori utilizzano versioni differenti di WoW64. Ad esempio, la versione di Windows a 64 bit sviluppata per l'architettura IA-64 impiega WoW64 emulare le istruzioni x86. Questa emulazione richiede un maggior numero di risorse rispetto a WoW64 per x86-64, poiché il sistema deve passare dalla modalità a 64 bit alla modalità compatibile durante l'esecuzione di programmi a 32 bit. WoW64 su x86-x64 invece non richiede l'emulazione di istruzioni: in questo caso il sottosistema WoW64 emula solo l'ambiente a 32

bit con un layer supplementare tra l'applicazione a 32 bit e le API di Windows a 64 bit, con un peggioramento delle prestazioni di circa il 2% [KAR]. La compilazione del codice a 64 bit non solo consente di evitare il layer di emulazione ma fornisce anche un ulteriore guadagno di prestazioni, rese possibili dalle modifiche architettoniche del microprocessore (come l'aumento dei numero di registri *general-pourpose*). Mediamente per un programma ci si può aspettare un miglioramento delle prestazioni del 5-15% [KAR] con una semplice ricompilazione. Il sottosistema WoW64 isola i programmi a 32 bit da quelli a 64 bit reindirizzando le chiamate ai file e al registro, prevenendo l'accesso da parte dei programmi a 32 bit ai dati di quelli a 64 bit. Questo isolamento consente di evitare errori di compatibilità: le applicazioni a 32 bit necessiteranno librerie della medesima tipolgia.

3.2 GNU/Linux

Nel sistema operativo GNU/Linux sono disponibili molti pacchetti compilati in modo nativo AMD64 a 64 bit. Ne esistono però alcuni, in particolare le applicazioni di terze parti *closed-source*, che non lo sono. Queste ultime non potranno essere eseguite direttamente nei processori x86-64.

Esistono tuttavia diversi metodi per superare questo problema: il dual-boot, la virtual machine, il chroot a 32 bit e la IA-32 Suite.

3.2.1 Dual boot / VM

Un metodo poco pratico ed inefficiente per eseguire applicazioni a 32 bit su un sistema a 64 bit è il dual boot del sistema tramite l'utilizzo di due sistemi operativi a 32 e 64 bit in una installazione *side-by-side*. Risolve il problema ma rimane poco pratico.
Un approccio migliore è quello di installare ed eseguire un sistema completo a 32 bit su di un sistema virtuale come *Qemu* o *VMWare*. Come il *dual boot*, questo metodo è lento.

3.2.2 Chroot

Il processore x86-64 è in grado di passare rapidamente tra le modalità a 32-bit e 64-bit, quindi piccole applicazioni statiche a 32 bit dovrebbero funzionare perfettamente su un sistema a 64 bit senza modificare nè il sistema o l'applicazione.
La maggior parte delle applicazioni moderne si collegano dinamicamente a una o più librerie ed è essenziale che tali applicazioni riescano a collegarsi a librerie del loro stesso tipo, altrimenti il programma non verrà eseguito.
Questo si può ottenere installando un sistema a 32 bit, completo di tutte le librerie e le applicazioni, senza kernel, in un chroot. Un chroot è un dispositivo software che crea una gabbia nel filesystem e le applicazioni in esecuzione all'interno di esso non possono muoversi al di fuori. È relativamente facile da realizzare, ma, come la soluzione della macchina virtuale e del *dual boot*, richiede molto spazio su disco ed è macchinoso se si necessita di eseguire una sola applicazione a 32 bit.

3.2.3 IA32 Suite

ia32-libs e *ia32-libs-multiarch* sono pacchetti che forniscono i moduli necessari alle applicazioni a 32 bit per essere eseguite correttamente in sistemi a 64 bit. Quando viene richiesto da un'applicazione a 32 bit l'accesso ad una libreria, il *dynamic-link library* ne fornisce la versione a 32 bit, se questa è disponibile, invece della versione a 64-bit delle applicazioni native. Sono inclusi molti moduli e viene richiesta una grande quantità di spazio su disco per l'installazione. Per questo motivo (e per l'esistenza quasi onnipresente di versioni a 64 bit per la maggior parte delle applicazioni), si ha il desiderio di eliminare gradualmente ia32-libs e ia32-libs-multiarch.
In un sistema operativo a 64 bit, può essere ancora necessario eseguire un'applicazione a 32 bit, quindi ia32-libs-multiarch può tornare ancora utile.

3.2.4 Linux

Seguendo l'esempio dagli sviluppatori Sparc, tutte le piattaforme con doppia architettura (32 e 64 bit) installano le librerie a 64 bit in percorsi che terminano con `/lib64`, ad esempio `/usr/X11R6/lib64` ([JAEG]). Il *linker* dinamico a 64 bit è configurato per ricercare questi percorsi. Per le librerie a 32 bit, invece, non è intercorsa alcuna modifica.
Questa impostazione ha il vantaggio che i pacchetti costruiti per la piattaforma a 32 bit possono essere installati senza alcuna modifica; infatti per questi ultimi tutto resta uguale alla piattaforma a 32 bit. I programmi a 64 bit necessitano di un po' più di lavoro in quanto spesso gli script di configurazione ricercano direttamente i percorsi di libreria trovando solo la libreria a 32 bit: utilizzano ad esempio `/usr/lib` o hanno makefile con percorsi esplicitati. Gli script di configurazione generati da GNU `autoconf` offrono un'opzione per specificare direttamente il percorso di installazione della libreria. Anche `ldconfig` gestisce librerie sia a 32 che 64 bit sia nel file di configurazione (`/etc/ld.so.conf`) che nel file di cache (`/etc/ld.so.cache`). `ldconfig` registra le librerie a 64 bit nella propria cache in modo che il linker dinamico possa facilmente trovare le librerie a 32 e 64 bit.

Compatibilità Cpu a 64 bit

Processore	64	64	64	64
Sistema Operativo	64	64	32	32
Programma Applicativo	64	32	32	64
	Ok	Ok	Ok	No

Tabella 3.1: Funzionalità nelle diverse architetture

Capitolo 4

Porting di sistemi a 64 bit

4.1 Pro e contro

La convenienza di una migrazione del sorgente a 64 bit, come con qualsiasi altro problema di prestazione, dipende dalla particolare situazione. In ogni caso, si dovrebbero valutare attentamente i pro e i contro.

4.1.1 Pro

- i processi a 64 bit hanno un maggiore spazio di indirizzi
- matematica ottimizzata a 64 bit,
- il kernel 64 bit di un sistema operativo utilizza una maggiore quantità di memoria disponibile per migliorare molti aspetti del lavoro.

4.1.2 Contro

- è necessaria una maggiore memoria per molte operazioni (i puntatori occupano una dimensione maggiore, soprattutto nei sorgenti che contengono riferimenti in tutto il codice),
- minor efficienza del processore (se confrontiamo le modalità a 32 bit e 64 bit),

- la dimensione del codice aumenta anche a causa dei prefissi ed istruzioni ulteriori contenenti operandi ad 8 byte invece di quelli a 4 byte.

In molti casi, i vantaggi superano gli svantaggi di cui sopra; ad esempio, molte applicazioni raggiungono il limite di memoria disponibile. L'aritmetica a 64 bit, inoltre, offre un notevole miglioramento delle prestazioni per alcune applicazioni; ad esempio, quelle che lavorano con la grafica, codifica video, giochi, ecc.

4.2 Piattaforma comune

Lo sviluppo di due diverse modalità (32 bit e 64 bit) genera complessità nei programmi e aumenta i costi dei test necessari al loro debugging. È necessario assicurarsi che siano presenti le due versioni e, soprattutto, che la versione necessaria venga selezionata automaticamente. Di solito si può risolvere questo problema piuttosto semplicemente attraverso le funzioni del sistema operativo di re-indirizzamento.
In XCModel si è voluto realizzare un codice sorgente che fosse compilabile sia a 32 che a 64 bit.

4.3 Stima del costo della migrazione

Prima di iniziare a progettare la migrazione di un progetto per un sistema a 64 bit, sarebbe opportuno essere in grado di stimare la quantità di lavoro ed il costo degli strumenti da utilizzare. Bisognerebbe includere almeno i seguenti componenti che costituiscono la base per poter realizzare una migrazione a 64 bit:

1. hardware e software a 64 bit
2. compilatore per applicazioni a 64 bit
3. versioni a 64 bit delle librerie

4. formazione del personale e l'acquisto di strumenti aggiuntivi
5. modifica del codice
6. adattamento del sistema di test
7. protezione delle unità di software

In XCModel molte delle voci precedenti sono già disponibili gratuitamente,in quanto utilizza software free. Non sono presenti inoltre né sistemi di test né protezione del software.

Capitolo 5

Analisi del software

La programmazione, specialmente in C, è resa difficoltosa dal linguaggio che è di tipo *low-level* e fornisce una scarsa protezione contro i più comuni errori di programmazione.
Al crescere della complessità dei programmi, si rendono necessari dei tool che ne possano incrementare la qualità, in particolare la correttezza e la velocità.

5.1 Analisi statica ed analisi dinamica

Possiamo suddividere l'analisi di un programma in due gruppi in base al momento in cui essa viene eseguita: l'analisi statica e l'analisi dinamica.
L'analisi statica comprende l'analisi del codice sorgente o del codice macchina senza la necessità che venga eseguito.
Molti tool eseguono analisi statica in particolare i compilatori. Esempi di analisi statica eseguita dai compilatori includono l'analisi di correttezza, come il *type checking* e l'analisi per l'ottimizzazione che individua le modifiche necessarie per un miglioramento delle performance.
Il vantaggio principale degli analizzatori codice statico è la possibilità di ridurre notevolmente i costi per l'eliminazione difetti in un programma. Infatti quanto prima viene rilevato un errore, meno costoso è correggerlo. La correzione di un errore nella fase di test del codice è cinque volte più costosa che

nella fase di progettazione [MCCON].
L'analisi dinamica analizza invece un programma durante la sua esecuzione. Sono molti gli strumenti che effettuano una analisi dinamica: i profiler, i checker ed i visualizzatori di esecuzione.
I tool che eseguono analisi dinamica devono aggiungere del codice al programma analizzato, di solito chiamato *instrumentation code*, che può essere inserito inline o con routine esterne. Il codice di analisi viene eseguito come parte del programma da analizzare, non influenzandone l'esecuzione (magari rallentandola), ed esegue delle operazioni aggiuntive come la misura delle performance o l'identificazione di bug.
I due approcci sono complementari: l'analisi statica considera tutti percorsi di esecuzione in un programma, mentre l'analisi dinamica prende in considerazione un singolo percorso. L'analisi dinamica è tipicamente più precisa di quella statica poiché lavora con valori generati durante l'esecuzione; motivo per cui l'analisi dinamica è spesso meno complessa di quella statica.
Sono diversi i vantaggi che offre l'analisi statica che la rendono il metodo più appropriato per rilevare gli errori nel codice a 64 bit:

- è possibile controllare l'intero codice del programma: possono essere testati frammenti di codice che vengono eseguiti molto raramente riuscendo ad offrire una copertura del codice quasi completa.
- è scalabile: consente di analizzare sia un piccolo che un grande progetto con la stessa semplicità.
- si possono notare eventuali problemi anche senza conoscere tutte le peculiarità del codice.
- offre una diagnosi precoce degli errori.
- utilizzabile sia per il porting del codice in un sistema a 64 bit sia per lo sviluppo di nuovo.

Esistono anche altre tecniche per rilevare errori nei sorgenti di un programma da prendere in considerazione:

5.2 La revisione del codice

Il metodo più provato ed affidabile per la ricerca degli errori è la revisione del codice [MCCON]. Si basa sulla lettura contemporanea del codice da parte di diversi sviluppatori seguendo alcune regole.

Purtroppo, è inapplicabile su larga scala a causa delle dimensioni dei programmi attuali e di XCModel in particolare. Può essere considerato, piuttosto, un buon mezzo per evitare errori in un sorgente in fase di sviluppo.

5.3 White box

Il metodo white box cerca di eseguire il massimo numero possibile di rami di codice utilizzando un debugger o altri strumenti. Una maggiore copertura del codice durante l'analisi rende il test maggiormente esaustivo.
Attualmente è impossibile testare completamente l'intero codice di un programma con il metodo white box viste le enormi dimensioni delle applicazioni. Ad oggi il metodo *white box* viene utilizzato quando viene rilevato un errore e si vuole scoprire cosa lo abbia causato.

Il debug *step-by-step* è improponibile per effettuare il debug di applicazioni che elaborano dati di grandi dimensioni, visto che potrebbe richiedere molto tempo: si dovrebbero prendere in considerazione in questo caso l'utilizzo di sistemi di registrazione.

5.4 Black Box (unit-test)

Il principio di funzionamento di questa tecnica risiede nello scrivere una serie di unit-test per le funzioni che si vogliono controllare. Le funzioni devono essere considerate *black box* poiché gli unit-test non tengono conto dell'organizzazione interna di una funzione.

Questo punto di vista è supportato dal fatto che i test sono sviluppati prima che le funzioni vengano scritte fornendo così un maggiore controllo sulla loro funzionalità in termini di specifica.

Gli unit-test si sono dimostrati efficaci nello sviluppo di progetti semplici e complessi. Uno dei vantaggi degli unit-test è il poter controllare la correttezza delle modifiche introdotte nel programma durante il processo di sviluppo. Il fatto che i test vengano eseguiti in breve tempo consente allo sviluppatore, che ha modificato il codice, di rilevare un eventuale errori e correggerli immediatamente.

5.5 Testing manuale

Questo metodo può essere considerato l'ultimo passo di un processo di sviluppo. Il test manuale esiste perché è impossibile rilevare tutti gli errori in modo automatico o con la revisione del codice.

Non si dovrebbe fare affidamento solo su uno dei metodi menzionati, anche se l'analisi statica si è rivelata la tecnica più efficace di rilevare errori a 64 bit.

5.6 Tool

Una parte consistente di tempo è stata impiegata nella ricerca di tool che si potessero adattare alla particolare struttura del progetto XCModel. I tool che ho scelto di utilizzare durante questo lavoro di tesi, oltre ai debugger come *gdb*, sono i seguenti:

5.6.1 Valgrind

Valgrind è un tool per il debug di memoria, rileva lo spreco di memoria ed effettua il profiling. Prende il nome dall'ingresso principale al Valhalla nella mitologia norrena. Valgrind è stato originariamente progettato per essere un tool per il debug di memoria su GNU/Linux x86. Da allora si è evoluto fino a diventare un framework generico per la creazione di strumenti di analisi dinamica, come checker e profiler, venendo impiegato da un vasto numero di

progetti basati su GNU/Linux. Valgrind è in sostanza una macchina virtuale che utilizza tecniche di compilazione just-in-time (JIT), tra cui la ricompilazione dinamica. Nessuna parte del programma originale viene eseguita direttamente. Valgrind prima traduce il programma in una forma temporanea, chiamata Rappresentazione Intermedia (IR), indipendente dall'architettura, poi lo porta in una forma basata su SSA.

5.6.2 Memcheck

Memcheck è lo strumento predefinito di Valgrind. Rileva e riporta un numero di errori della memoria difficili da diagnosticare, come ad esempio un accesso della memoria non previsto, l'uso di valori non inizializzati o non definiti, memoria heap resa disponibile non correttamente, puntatori sovrapposti e perdite di memoria. Utilizzando Memcheck i programmi verranno eseguiti da dieci a trenta volte più lentamente rispetto alla loro normale esecuzione. Memcheck si limita a riportare gli errori e non ne impedisce la generazione. Di seguito un esempio del risultato prodotto da *memcheck*:

```
==18671== Invalid read of size 4
==18671==    at 0x4D22EF: wireframe_do (visual.c:350)
==18671==    by 0x4D5116: Trim_visual_do (visual.c:867)
==18671==    by 0x44804F: renderSurfaces (model.c:411)
==18671==    by 0x4694BC: renderScena (modif.c:735)
==18671==    by 0x46CEBD: opzTrasformSup (modifCP.c:260)
==18671==    by 0x4686A1: modify (modif.c:362)
==18671==    by 0x4AF51F: hierInteractiveButton (Win_Hierarchical.c:679)
==18671==    by 0x4C971D: BRManager (eventman.c:364)
==18671==    by 0x4C9E88: EventManager (eventman.c:559)
==18671==    by 0x4C9C68: EventManagerMainLoop (eventman.c:504)
==18671==    by 0x4CC6FC: RunGUI (gui.c:139)
==18671==    by 0x41A4EC: main (Main.c:511)
==18671==  Address 0xffffffffe1715a840 is not stack'd, malloc'd or (recently)
free'd
==18671==
==18671==
==18671== Process terminating with default action of signal 11 (SIGSEGV)
==18671==  Access not within mapped region at address 0xFFFFFFFE1715A840
==18671==    at 0x4D22EF: wireframe_do (visual.c:350)
==18671==    by 0x4D5116: Trim_visual_do (visual.c:867)
==18671==    by 0x44804F: renderSurfaces (model.c:411)
==18671==    by 0x4694BC: renderScena (modif.c:735)
==18671==    by 0x46CEBD: opzTrasformSup (modifCP.c:260)
==18671==    by 0x4686A1: modify (modif.c:362)
==18671==    by 0x4AF51F: hierInteractiveButton (Win_Hierarchical.c:679)
```

```
==18671==    by 0x4C971D: BRManager (eventman.c:364)
==18671==    by 0x4C9E88: EventManager (eventman.c:559)
==18671==    by 0x4C9C68: EventManagerMainLoop (eventman.c:504)
==18671==    by 0x4CC6FC: RunGUI (gui.c:139)
==18671==    by 0x41A4EC: main (Main.c:511)
==18671==  If you believe this happened as a result of a stack
==18671==  overflow in your program's main thread (unlikely but
==18671==  possible), you can try to increase the size of the
==18671==  main thread stack using the --main-stacksize= flag.
==18671==  The main thread stack size used in this run was 8388608.
==18671==
==18671== HEAP SUMMARY:
==18671==     in use at exit: 1,731,116 bytes in 3,786 blocks
==18671==   total heap usage: 69,716 allocs, 65,930 frees, 57,774,552 bytes
allocated
==18671==
==18671== 1 bytes in 1 blocks are still reachable in loss record 1 of 1,627
==18671==    at 0x13B7D35B: malloc (vg_replace_malloc.c:270)
==18671==    by 0x14C88D91: strdup (strdup.c:42)
==18671==    by 0x4CC79A: AddLabel (labels.c:54)
==18671==    by 0x4AC35F: CreaWindowWMess_Err (Win_Errors.c:90)
==18671==    by 0x4198A1: CreateAllWindows (Main.c:259)
==18671==    by 0x41A4BE: main (Main.c:505)
```

5.6.3 Cppcheck

Cppcheck è un tool per l'analisi statica di codice C e C ++. È uno strumento versatile che può controllare il codice non standard. Supporta una vasta gamma di controlli statici che non possono essere svolti dal compilatore stesso che vengono eseguiti a livello di codice sorgente. Alcuni dei controlli supportati includono:

- controllo di variabili automatiche,
- controllo dei limiti degli array,
- controllo di classi (ad esempio funzioni inutilizzate, inizializzazione delle variabili e la duplicazione della memoria),
- l'utilizzo di funzioni deprecate o sostituite secondo l'Open Group,
- controllo di sicurezza delle eccezioni, per esempio l'utilizzo di allocazione di memoria e controlli distruttori,
- le perdite di memoria, ad esempio a causa di perdita di scope senza de-allocazione,

- perdite di risorse, ad esempio a causa di un file handler non chiuso,
- utilizzo non valido di funzioni della Standard Template Library,
- vari errori stilistici e prestazionali.

Un esempio del risultato prodotto da *cppcheck*:

```
[hrayt/load_model.c:648]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:651]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:655]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:664]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:674]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:685]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:690]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:695]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:700]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:707]: (warning) scanf without field width limits can crash
with huge input data.
[hrayt/load_model.c:722]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:731]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/load_model.c:368]: (error) Resource leak: nfp
Checking hrayt/load_model.c: DEFINEH...
24/227 files checked 5% done
Checking hrayt/lzwio.c...
[hrayt/lzwio.c:99]: (style) The scope of the variable 'i' can be reduced.
[hrayt/lzwio.c:58]: (style) Checking if unsigned variable 'colnum' is less than
zero.
Checking hrayt/lzwio.c: DEFINEH...
25/227 files checked 5% done
Checking hrayt/parameters.c...
[hrayt/parameters.c:82]: (portability) scanf without field width limits can
crash with huge input data on some versions of libc.
[hrayt/parameters.c:87]: (warning) scanf without field width limits can crash
with huge input data.
[hrayt/parameters.c:154]: (warning) scanf without field width limits can crash
with huge input data.
[hrayt/parameters.c:194]: (warning) scanf without field width limits can crash
with huge input data.
Checking hrayt/parameters.c: DEFINEH...
26/227 files checked 5% done
Checking hrayt/primitives.c...
[hrayt/primitives.c:53]: (error) Uninitialized variable: point
```

```
[hrayt/primitives.c:53]: (error) Uninitialized variable: delta
Checking hrayt/primitives.c: DEFINEH...
27/227 files checked 5% done
Checking hrayt/ray_trace.c...
[hrayt/ray_trace.c:322]: (style) The scope of the variable 't' can be reduced.
[hrayt/ray_trace.c:322]: (style) The scope of the variable 't1' can be reduced.
[hrayt/ray_trace.c:322]: (style) The scope of the variable 's' can be reduced.
[hrayt/ray_trace.c:323]: (style) The scope of the variable 'res' can be reduced.
[hrayt/ray_trace.c:429]: (style) The scope of the variable 'gc' can be reduced.
[hrayt/ray_trace.c:429]: (style) The scope of the variable 'g_c' can be reduced.
[hrayt/ray_trace.c:429]: (style) The scope of the variable 'g' can be reduced.
[hrayt/ray_trace.c:371]: (error) Uninitialized variable: refr_delta
[hrayt/ray_trace.c:398]: (error) Uninitialized variable: refr_delta
Checking hrayt/ray_trace.c: DEFINEH...
28/227 files checked 6% done
```

5.6.4 Callgrind

Callgrind è uno strumento di profiling che registra la cronologia delle chiamate tra le funzioni di un programma in esecuzione, come in un *call-graph*. Per impostazione predefinita i dati raccolti sono costituiti:

- dal numero di istruzioni eseguite,
- dal loro rapporto con le linee di sorgente,
- dal rapporto del *caller/callee* tra le funzioni,
- dal numero di tali chiamate.

La simulazione di cache e/o branch prediction (simile a Cachegrind) è in grado di fornire ulteriori informazioni sul comportamento dell'applicazione.

Per la rappresentazione dei dati ed il controllo interattivo del profiling, sono previsti due strumenti a linea di comando:

callgrind_annotate : legge i dati di profilo e stampa una lista ordinata di funzioni,

callgrind_control questo comando consente di osservare e controllare in modo interattivo lo stato di un programma in esecuzione sotto il con-

trollo di Callgrind senza fermarlo. È possibile ottenere informazioni statistiche e l'analisi dello stack corrente.

Per la visualizzazione grafica dei dati, ho utilizzato KCachegrind, una GUI basata su KDE / Qt che rende facile navigare tra la grande quantità di dati che Callgrind produce.

5.6.5 Frama-c

Frama-C (*Framework for Modular Analysis of C programs*) è una piattaforma dedicata all'analisi del codice sorgente scritto in C. È un framework collaborativo ed estensibile che comprende diverse tecniche di analisi. questo approccio permette all'analizzatore di riutilizzare i risultati ottenuti dagli altri plugin presenti nel framework.
Frama-C ha un'architettura modulare di tipo plugin e si basa sul CIL (C Intermediate Language) per generare un albero di sintassi astratta che supporta le annotazioni scritte in ACSL (ANSI / ISO C Specification Language). I diversi moduli possono manipolare l'albero di sintassi astratta per aggiungere annotazioni ACSL.
I plugin più frequentemente utilizzati sono:

value analisys : calcola un valore o un insieme di possibili valori per ogni variabile presente in un programma. Utilizza la tecnica dell'interpretazione astratta e molti altri plugin fanno uso dei suoi risultati.

jessie : verifica le proprietà in modo deduttivo.

impact analisys : evidenzia nel codice sorgente C gli impatti di una modifica.

slicing : questo plugin permette di suddividere un programma generando un programma C di minori dimensioni che conserva le proprietà indicate.

spare code : rimuove il codice inutile da un programma C.

Esempio di output:

```
sources/xcsurf/Action_Hierarchical.c:492:[kernel] warning: Floating-point
constant 0.0000009 is not represented exactly. Will use 0x1.e32f0ee144531p-21.
See documentation for option -warn-decimal-float
sources/include/xtools/xtools.h:43:[value] warning: during initialization of
variable 'dispGUI', size of type 'struct _XDisplay'
sources/include/xtools/xtools.h:45:[value] warning: during initialization of
variable 'gcGUI', size of type 'struct _XGC'
sources/xcsurf/Main.c:152:[kernel] warning: Neither code nor specification for
function InitTable, generating default assigns from the prototype
sources/xcsurf/Main.c:155:[kernel] warning: Neither code nor specification for
function Initgraph, generating default assigns from the prototype
sources/xcsurf/graph.c:60:[kernel] warning: Neither code nor specification for
function XOpenDisplay, generating default assigns from the prototype
sources/xcsurf/graph.c:61:[kernel] warning: out of bounds read.
sources/xcsurf/graph.c:62:[kernel] warning: out of bounds read.
sources/xcsurf/graph.c:62:[kernel] warning: out of bounds read.
sources/xcsurf/graph.c:63:[kernel] warning: out of bounds read.
sources/xcsurf/graph.c:63:[kernel] warning: out of bounds read.
sources/xcsurf/graph.c:64:[kernel] warning: out of bounds read.
sources/xcsurf/graph.c:64:[kernel] warning: out of bounds read.
sources/xcsurf/graph.c:64:[kernel] warning: out of bounds read.
sources/xcsurf/graph.c:65:[kernel] warning: Neither code nor specification for
function XCreateGC, generating default assigns from the prototype
sources/xcsurf/graph.c:66:[kernel] warning: out of bounds read.
sources/xcsurf/graph.c:66:[kernel] warning: out of bounds read.
```

Value analysis

Questa analisi che viene attivata tramite l'opzione -val:

Tipologia dei valori delle variabili intere:

- insieme finito: $a \in 4;5;$.
- intervallo: $i \in [0..100]$ $a \in [x..y]$.
- intervallo periodico: $i \in [2..42]\%2, 10$.
- tutti i valori dell'intervallo in cui i resti della divisione per 10 sia 2: 2, 12, 22, 32, 42.

Tipologia dei valori delle variabili floating point:

- valore esatto (3.0).
- intervallo ($[-1.0..1.0]$).

Array e puntatori:
insieme di indirizzi: base + offset (in bytes)

- $p \in \{\&a; \&b; \}$.

- $p \in \{\&t + \{0; 4; 8; 12; \}; \&s + \{0; \}; \}$.

Allarmi generati:

- divisione per 0 e modulo 0: $10/y$ $10\%y$.

- shift indefinito (ad es. fuori da [0..31]) `1` $\ll$ `c`.

- overflow con aritmetica con segno o senza segno.

- utilizzo pericoloso di float (operazioni che generano infinito o NaN, utilizzo di interi come float.

- variabili non inizializzate e puntatori a variabili locali.

- accesso alla memoria non consentito: accesso a vettori fuori dai limiti.

- comparazione pericolosa dei puntatori ed effetti collaterali.

Esempio di output:

```
  shmodel ? {0}
  lightpos ? {0}
  S_dispGUI[...] ? [--..--] or UNINITIALIZED
  S_gcGUI[...] ? [--..--] or UNINITIALIZED
[value] computing for function InitAll <- main.
        Called from sources/xcsurf/Main.c:436.
sources/xcsurf/Main.c:141:[value] entering loop for the first time
sources/xcsurf/Main.c:141:[value] assigning non deterministic value for the
first time
[value] computing for function InitTable <- InitAll <- main.
        Called from sources/xcsurf/Main.c:149.
sources/xcsurf/Main.c:149:[kernel] warning: Neither code nor specification for
function InitTable, generating default assigns from the prototype
[value] using specification for function InitTable
[value] Done for function InitTable
[value] computing for function InitTable <- InitAll <- main.
        Called from sources/xcsurf/Main.c:150.
[value] Done for function InitTable
[value] computing for function Initgraph <- InitAll <- main.
        Called from sources/xcsurf/Main.c:152.
sources/xcsurf/Main.c:152:[kernel] warning: Neither code nor specification for
function Initgraph, generating default assigns from the prototype
[value] using specification for function Initgraph
[value] Done for function Initgraph
[value] computing for function Init <- InitAll <- main.
        Called from sources/xcsurf/Main.c:153.
[value] computing for function XOpenDisplay <- Init <- InitAll <- main.
```

5.6.6 Pahole

Pahole è uno strumento che aiuta a scoprire lo speco di spazio nelle strutture dati. Nella distribuzione Debian viene fornito dal pacchetto *dwarves*. Pahole analizza il file binario prodotto da *GCC* (con l'opzione -g per abilitare i simboli di debug) ed elenca le strutture dati (struct, union) che contengono uno spreco di spazio, come nell'esempio seguente:

```
struct lb_t {
        XPoint                  pos;                /*     0     4 */
        /* XXX 4 bytes hole, try to pack */
        long unsigned int       col;                /*     8     8 */
        char *                  caption;            /*    16     8 */
        int                     nameid;             /*    24     4 */
        int                     enabled;            /*    28     4 */
        struct lb_t *           next;               /*    32     8 */
        /* size: 40, cachelines: 1, members: 6 */
        /* sum members: 36, holes: 1, sum holes: 4 */
        /* last cacheline: 40 bytes */
};
```

la struttura di dimensione 36 byte in realtà ne occupa 40 a causa dell'allineamento del primo elemento. In figura 5.1 viene evidenziato graficamente.

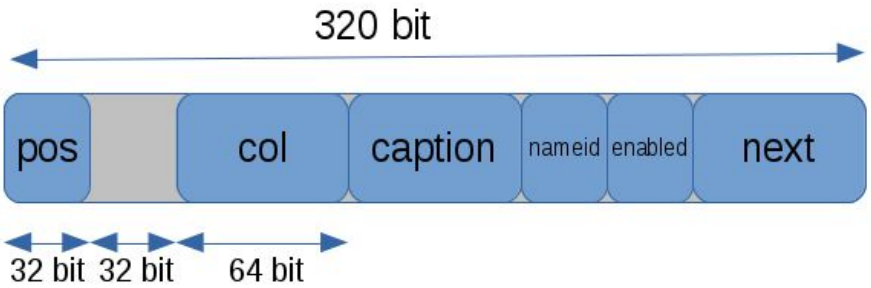

Figura 5.1: Esempio di inefficienza di spazio in una struttura

Riorganizzando gli elementi della struttura si può correggere lo spreco di spazio (figura 5.2).

```
struct lb_t {
        char *                  caption;        /*      8 */
        struct lb_t *           next;           /*      8 */
        long unsigned int       col;            /*      8 */
        int                     nameid;         /*      4 */
        int                     enabled;        /*      4 */
        XPoint                  pos;            /*      4 */
};
```

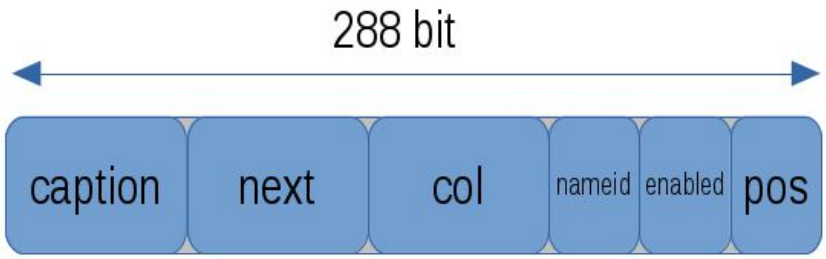

Figura 5.2: La struttura riorganizzata

Capitolo 6

Errori a 64 bit

Alcuni linguaggi di alto livello non sono influenzati dal cambiamento delle dimensioni dei tipi, altri invece come il C, C ++, Objective C e assembler possono esserlo.
Il tempo richiesto per il porting di un'applicazione da 32 bit a 64 bit può essere significativo, a seconda di come l'applicazione è stata scritta e mantenuta.
Quando parliamo di errori 64 bit, si intendono quei casi in cui un frammento di codice che funziona bene nella versione a 32 bit di un'applicazione causa errori dopo la ricompilazione nella modalità a 64 bit. Gli errori a 64 bit si verificano più frequentemente nei seguenti tipi di frammenti di codice:

- codice basato su ipotesi errate circa le dimensioni di tipo (per esempio, presuppone che la dimensione del puntatore sia sempre 4 byte);
- elaborazione di codice con grandi array la cui dimensione è superiore a 2 Gbyte su sistemi a 64 bit;
- codice responsabile per la scrittura dei dati e la lettura;
- codice contenente operazioni su bit;
- codice con aritmetica di indirizzi complessa;
- codice obsoleto;

In realtà, tutti gli errori che si verificano nel codice, quando viene ricompilato per i sistemi a 64-bit, nascono dal non rispetto del C standard.

6.1 int e long

Nelle piattaforme 32 bit la dimensione dei tipi int e long sono le stesse rendendo intercambiabili i due tipi di dati. Nei 64 bit invece il tipo long è di dimensioni maggiori. Ad esempio un puntatore non può essere memorizzato in una variabile di tipo int ma nel tipo long (in Unix) o meglio nel tipo *intptr_t*.
Le costanti intere senza tipo (unsigned int) possono condurre ad un troncamento inaspettato come ad esempio nel seguente codice `long t = 1 << a;`. In entrambi i sistemi a 32 e 64 bit il massimo valore di `a` sarà 31 in quanto il tipo `1<<a` è un int. Per ottenere uno shift nei 64 bit è necessario utilizzare un long e cioè 1L.
I tipi degli identificatori di un *enum* vengono definiti dall'implementazione ma tutte le costanti sono dello stesso tipo. Ad esempio GCC le imposta ad int.

6.2 Aritmetica floating point

Lo standard IEEE 754 definisce che le operazioni floating-point di base devono essere esatte, ma il risultato di alcuni algoritmi può variare tra le diverse architetture. Il problema compare con le operazioni di tipo float e double e risiede nel fatto che nell'architettura x86 queste operazioni vengono valutate dalla FPU x87 in precisione long double. Il compilatore può decidere di lasciare i risultati intermedi nella FPU x87 (con un tipo long double) o di convertirli nel tipo di destinazione. In base a quando questa conversione viene effettuata si verificano differenti errori di arrotondamento. Ad esempio:

```
float b, c;
b = 1 / 3.0f;
c = b * 3.0f - 1.0f;
printf ("c: %.20f\n", c);
```

questo programma fornirà differenti risultati su una macchina Linux/AMD64 con eseguibili a 32 e 64 bit. A 32 bit il risultato sarà 0.00000002980232238770 mentre a 64 bit: 0.00000000000000000000.
Verrà restituito lo stesso risultato invece se compilato con ottimizzazioni, poiché senza ottimizzazioni b viene memorizzato in memoria come tipo float mentre con le ottimizzazioni b viene lasciato nella FPU. ISO C99 definisce la macro `FLT_EVAL_METHOD` nell'header `<float.h>` per questo caso.

FLT_EVAL_METHOD	**float**	**double**	**long double**
0	float	double	long double
1	double	double	long double
2	long double	long double	long double

Tabella 6.1: Macro FLT_EVAL_METHOD

Viene impostata come indicato nella tabella 6.1 a 0 se la valutazione viene effettuata nel range e nella precisione del tipo (è il sistema più diffuso sui sistemi GNU/Linux). Ad 1 se la valutazione dell'espressione del tipo float e double è nel range e precisione del double ed il long double nel range e precisione del long double. A 2 se tutte le valutazioni vengono effettuate nel range e precisione del long double. Quest'ultimo è il valore sulle macchine GNU/Linux x86. A -1 se indeterminato.
Questo problema della valutazione dei floating point non è riconducibile direttamente ai 64 bit, ma piuttosto tra il codice x86 e le altre piattaforme come AMD64.

6.3 Puntatori

Durante il porting del codice per sistemi a 64 bit, un errore comune riscontrato nel codice di XCModel è rappresentato dalle variazioni delle dimensioni dei puntatori e la dimensione degli interi. In un ambiente che utilizza il modello di dati ILP32 gli interi ed i puntatori hanno la stessa dimensione. I puntatori sono spesso espressi come int, unsigned, long, ed altri tipi inappropriati. Si dovrebbero utilizzare esclusivamente tipi di dato memsize per la rappresentazione di puntatori.

Di seguito si possono notare alcuni errori riscontrati nel sorgente di XCModel:

```
SetCheckPointOn (window,1,(int) &p1);
```

dove viene effettuato un cast ad int del puntatore p1.

```
((hierResultObject->hier).Ow)=(float**)malloc(sizeof(float)*(m[sup2]+1));
```

dove viene effettuata una operazione di *sizeof()* ad un tipo *float* anziché *float**. Nei 32 bit funzionava correttamente, nei 64 no.

I due esempi precedenti sono pericolosi poiché l'errore nel programma potrebbe rimanere silente per diverso tempo.
In un sistema a 64 bit può funzionare correttamente con piccole quantità di dati, ovvero finché gli indirizzi rimangono all'interno dei primi 4 Gbyte di memoria. Oltre questo limite si genererà un overflow conducendo il programma ad un comportamento imprevedibile.

In Windows quando si necessita di un tipo di puntatore a 32-bit, ad esempio per utilizzare funzioni API obsolete, si deve ricorrere a funzioni speciali come *LongToIntPtr*, *PtrToUlong*, etc.

6.4 Dimensioni delle struct

Le strutture dati del linguaggio C sono allineate in modo tale da rendere il loro accesso più efficiente, infatti il compilatore C lascia spazi vuoti tra i campi delle *struct* per allinearle agli indirizzi delle word e quindi accelerare l'accesso ad esse. È possibile disattivare l'allineamento utilizzando speciali direttive *#pragma* per ridurre la quantità di memoria consumata. La quantità di memoria utilizzata però può spesso essere notevolmente ridotta cambiando semplicemente l'ordine dei campi nella struttura senza penalizzare le prestazioni.

Una crescita di dimensioni delle struct non è un errore, ma può portare ad consumo di una grande quantità di memoria e quindi ad una riduzione delle prestazioni. È una inefficienza del codice a 64 bit.

Un esempio:

```
struct lt_bak_t {
        char                    name[255];      /*     0   255 */
        /* XXX 1 byte hole, try to pack */
        /* --- cacheline 4 boundary (256 bytes) --- */
        short unsigned int      type;           /*   256     2 */
        /* XXX 6 bytes hole, try to pack */
        VEC_vector_t            location;       /*   264    24 */
        VEC_vector_t            direction;      /*   288    24 */
        VEC_vector_t            object_point;   /*   312    24 */
        /* --- cacheline 5 boundary (320 bytes) was 16 bytes ago --- */
        VEC_vector_t            view_up;        /*   336    24 */
        BOOLEAN                 vwupset;        /*   360     4 */
        Color_t                 light_color;    /*   364    12 */
        float                   intensity;      /*   376     4 */
        short int               concentration_exp;/*   380     2 */
        /* XXX 2 bytes hole, try to pack */
        /* --- cacheline 6 boundary (384 bytes) --- */
        VEC_real_t              max_range;       /*   384     8 */
        VEC_real_t              cone_cosine;     /*   392     8 */
        int                     flap_on;         /*   400     4 */
        /* XXX 4 bytes hole, try to pack */
        Plane_t                 flap;            /*   408    32 */
        /* size: 440, cachelines: 7, members: 14 */
        /* sum members: 427, holes: 4, sum holes: 13 */
        /* last cacheline: 56 bytes */
};
```

Tale struttura avrà 427 byte su un sistema a 32 bit. Ogni campo è allineato a 8 byte. Nella modalità di compilazione a 64 bit la struttura *lt_bak_t* avrà 440 byte. È un grande risparmio di memoria se dovessimo utilizzare, per esempio, milioni di elementi. Ciò che è più importante è che migliorerà le prestazioni.
Con poche strutture le dimensioni non contano, l'accesso sarà eseguito con la stessa velocità, ma quando sono presenti molti elementi il numero di accessi alla memoria diventa significativo.

Non è sempre possibile o conveniente cambiare l'ordine dei campi nelle struct. In generale, ogni campo è allineato all'indirizzo multiplo della dimensione di quel campo. Un campo di tipo *size_t* su un sistema a 64 bit viene allineato ad 8 byte, int a 4 byte, short a 2 byte, mentre i char non sono allineati. La dimensione di una tale struttura è allineata ad un multiplo della dimensione del suo elemento più grande. Ad esempio:

```
struct gui_t {
        Win_t *                     windows;      /*     0     8 */
        int                         status;       /*     8     4 */
        /* size: 16, cachelines: 1, members: 2 */
        /* padding: 4 */
        /* last cacheline: 16 bytes */
};
```

Gli elementi occuperanno 12 byte. Creando un vettore struct *gui_t[2]* la dimensione della struttura (12 byte), il campo windows della seconda struct verrà memorizzato su un indirizzo non allineato alla prima. Pertanto il compilatore dovrà aggiungere 4 byte vuoti alla prima struttura per rendere le sue dimensioni di 16 byte. Per ottimizzazione un campo in una struct è sufficiente disporre i campi in ordine di dimensione decrescente. I campi saranno così disposti senza spazi inutilizzati.

6.5 Allineamento dei dati

I processori riescono a lavorare in modo più efficiente quando i dati sono allineati correttamente, anzi, alcuni processori non possono lavorare con dati

non allineati.

I processori IA64 quando tentano di lavorare con dati non allineati generano un'eccezione; in particolare per utilizzare dati non allineati su Itanium è necessario dichiararlo esplicitamente al compilatore (ad esempio utilizzando la macro *UNALIGNED*). In questo caso il compilatore genera un codice speciale per trattare i dati non allineati. Non è molto efficiente, in quanto l'accesso ai dati sarà più lento.
Le eccezioni, invece, non vengono generate quando si utilizza dati non allineati su architettura x64, anche se si dovrebbe evitare; in primo luogo, perché l'accesso a questi dati è più lento, e in secondo luogo, perché in futuro si potrebbe portare il programma sulla piattaforma IA-64.

Tipi di dati	dimensione	allineamento	dimensione	allineamento	dimensione	allineamento	dimensione	allineamento
bool	1	1	1	1	1	1	1	1
wchar_t	2	2	2	2	4	4	4	4
short int	2	2	2	2	2	2	2	2
int	4	4	4	4	4	4	4	4
long int	4	4	4	4	4	4	8	8
long long int	8	8	8	8	8	4	8	8
float	4	4	4	4	4	4	4	4
double	8	8	8	8	8	4	8	8
long double	8	8	8	8	12	4	16	16
void*	4	4	8	8	4	4	8	8
	Windows 32 bit x86		Windows 64 bit x86-64		GNU/Linux 32 bit x86		GNU/Linux 64 bit x86-64	

Tabella 6.2: Tipi di dati: dimensioni ed allineamento

6.6 Union

La *union* è una struttura dati in cui i suoi membri occupano lo stesso spazio di memoria: sono sovrapposti. È possibile accedere ad un determinato spazio di memoria tramite un qualsiasi elemento della *union*. Utilizzando un puntatore come un numero intero, può essere conveniente usare una *union* ed utilizzare la rappresentazione numerica del tipo senza conversioni esplicite. Questo utilizzo è corretto per i sistemi a 32 bit e non corretto per quelli a 64 bit. È necessario utilizzare un tipo che corrisponda alle dimensioni del puntatore.Un altro modo usuale di utilizzare una *union* è quella di rappresentare un membro come insieme di diversi membri più piccoli.

6.7 Costanti numeriche

In un codice scritto male si possono spesso vedere costanti numeriche la cui sola presenza è pericolosa di per sé. Quando si effettua una migrazione di codice verso una piattaforma a 64 bit, queste costanti possono rendere il codice inefficiente se partecipano al calcolo degli indirizzi, alle dimensioni di un oggetto o alle operazioni su bit.

Esempi di costanti sono: il numero 4 usato per definire il numero di byte nel tipo di variabile, 32 a rappresentare il numero di bit nel tipo, 0X7FFFFFFF come massimo valore di variabili con segno o maschera per settare il bit più significativo a zero, 0X80000000 come minimo valore di variabile con segno o come maschera per selezionare il bit più significativo, 0XFFFFFFFF come massimo valore di una variabile unsigned o rappresentazione alternativa di -1 come indicatore di errore.

Mentre l'utilizzo di queste costanti è tollerato in un sistema a 32 bit, non lo è in un sistema a 64 bit in quanto può condurre a comportamenti impredicibili dell'applicazione.

6.8 Funzioni con un numero variabile di argomenti

Gli esempi maggiormente riportati sulla migrazione a 64 bit fanno riferimento ad un uso non corretto delle funzioni printf , scanf e delle loro varianti. L'uso improprio di funzioni con un numero variabile di argomenti è un errore comune, non solo per le architetture a 64 bit, ma per tutte le architetture. Esistono formati specifici per lavorare con i tipi memsize in funzioni come sscanf, printf. In Windows esiste il formato l, in Linux il formato z. Se si deve far migrare un codice che utilizza funzioni come sscanf, è possibile utilizzare macro speciali sotto forma di stringhe che si espandono nei formati necessari. La manipolazione dei puntatori utilizzando %X conduce ad un comportamento del programma non corretto di un sistema a 64 bit.
ISO C99 ha introdotto nuovi specificatori di formato per consentire la stampa e la scansione di alcuni tipi che potrebbero avere dimensioni dipendenti dall'architettura. Sono %p per stampare un valore di puntatore e il %Z per gli argomenti di tipo size_t. Un esempio:

```
void * p;
printf ("p ha valore% p e dimensione %Zd \n", p, sizeof (p));
```

6.9 Utilizzo di tipi dipendenti dalla dimensione

Alcune applicazioni richiedono dimensioni specifiche per i loro tipi di dati. ISO C99 ha introdotto un nuovo file di intestazione `stdint.h` che definisce i tipi di dati che hanno dimensioni predefinite e una corrispondente serie di macro. Vengono specificati anche i seguenti tipi:

- Tipi interi di esatta grandezza. I tipi interi con segno della forma *intN_t* (senza segno: *uintN_t*) di dimensione N sono definiti in generale 8, 16,

32, o 64 bit.

- Tipi interi di dimensioni minime. I tipi con segno *int_leastN_t* e senza segno *uint_leastN_t* sono definiti con una dimensione di almeno N bit (8, 16, 32 e 64).

- Tipi interi di dimensione minima fast. I tipi con segno *int_fastN_t* e senza segno *uint_fastN_t* hanno una dimensione di almeno N bit sono definiti come tipi interi fast con almeno quella dimensione (8, 16, 32 e 64).

- Tipi interi che memorizzano puntatori. Il tipo intero *intptr_t* e *uintptr_t* è capace di memorizzare un puntatore: una conversione tra un puntatore e questo tipo intero è sempre possibile.

- Tipi interi di dimensione massima. I tipi di interi *intmax_t* e *uintmax_t* possono memorizzare qualsiasi valore di tipo intero con segno o senza.

Si noti che una implementazione ISO C99 non richiede necessariamente tutti questi tipi. La libreria GNU C li implementa tutti e su tutte le architetture. In aggiunta sono state definite diverse macro per fornire i limiti dei diversi tipi. Ad esempio l'header `inttypes.h` definisce macro per i modificatori di formato sia per *printf* che *scanf*.

Capitolo 7

Prestazioni

Sono state condotte diverse prove per testare il funzionamento di XCModel in modalità 32 e 64 bit. Sono state condotte principalmente su due macchine: un notebook con processore Intel Pentium Dual Core T4500 ed un desktop con processore Intel Core i3 530. In entrambe le macchine sono state installate due versioni dello stesso sistema operativo; sul desktop è stata installata la distribuzione *GNU/Linux Mint Debian Edition* mentre sul notebook è stata scelta la *Opensuse* 13.2. Sono state condotte diverse sessioni tipo con XCModel anche tramite l'aiuto di script per l'automazione dell'input.
La compilazione del pacchetto è stata eseguita con l'opzione *-march=native* in modo che il compilatore *gcc* ottimizzi il codice per la specifica architettura hardware, sfruttando a pieno le caratteristiche del processore. Questi script fanno uso del pacchetto *xdotool*, un tool utilizzato per il test delle interfacce grafiche in ambiente GNU/Linux. È stato altresì inserita, nel codice di XCModel, una funzione che salva gli di eventi e li scrive su file. Questi eventi, tramite gli script di cui sopra, riproducono fedelmente il comportamento dell'utente sulle diverse piattaforme. È stato scelto questo approccio per testare il funzionamento del programma in ogni sua parte, interfaccia grafica compresa in quanto parte integrante e fondamentale dell'applicazione. Il programma così riprodotto viene analizzato dall'applicazione *callgrind* che inizia il processo di *profiling*. I risultati ottenuti vengono visualizzati

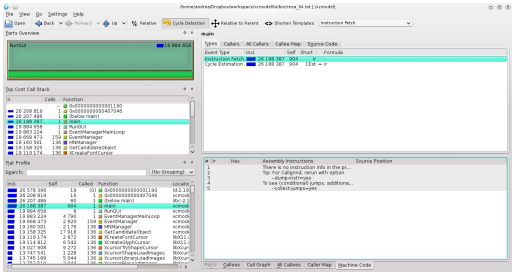

Figura 7.1: Test di resa su OpenSuse 32 bit

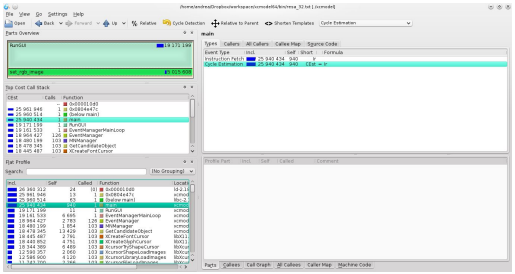

Figura 7.2: Test di resa su Opensuse 64 bit

all'utente tramite il pacchetto *kcachegrind* di cui un esempio in figura 7.1 ed in figura 7.2.

7.1 Risultati

I risultati ottenuti mostrano che XCModel non si avvantaggia della nuova architettura a 64 bit. Anzi, il codice a 32 bit viene eseguito più velocemente di circa l'1% a parità di processore.
La marcata differenza di prestazioni riscontrata nel passaggio da processori a 32 a quelli a 64 bit evidentemente risiede principalmente nella maggior potenza di calcolo degli stessi.
XCModel potrà trarre beneficio dalla nuova architettura principalmente dal maggior quantitativo di memoria che potrà utilizzare e dalla mancata obsolescenza che la progressiva dismissione delle librerie a 32 bit comporterà.

Conclusioni

Il lavoro di porting a 64 bit del pacchetto XCModel è stato un impegno lungo e complesso, ben più di quanto l'analisi iniziale del problema avesse lasciato intendere. Il codice è molto eterogeneo nella propria struttura, segno delle numerose modifiche apportate nel corso degli anni, riflettendo anche i diversi stili di programmazione lasciati nel tempo dal programmatore di turno.
Non è stato semplice, non solo riuscire ad apportare le correzioni necessarie, ma anche semplicemente capire quali strumenti utilizzare il controllo del codice.
È così iniziato un lungo percorso di studio delle problematiche più comuni nella migrazione a 64 bit ma soprattutto di ricerca degli strumenti e dei metodi adeguati al compito assegnato.
Già le prime analisi hanno rilevato che il sorgente del programma non era mai stato sottoposto ad un debug importante: assenza della maggior parte dei prototipi, chiamate a funzione con parametri errati, variabili dichiarate con tipo diverso nei vari moduli.
È stato comunque un lavoro interessante e stimolante che mi ha permesso di comprendere meglio una parte importante dello sviluppo di un software. Parte che spesso viene nascosta dai moderni sistemi di sviluppo integrato che relegano il programmatore a semplice *generatore* di codice.
Ho avuto piacere nel far parte di un progetto che si sta sviluppando nel corso degli anni e che riesce a coinvolgere così tante persone.

Sviluppi futuri

Per non vanificare il lavoro svolto in questa tesi, sarebbe opportuno applicare i metodi che ho descritto anche alle future revisioni del codice.
Infatti è già iniziato lo sviluppo della prossima generazione di architetture, quelle a 128 bit: gli sviluppatori della Microsoft Research stanno già lavorando sulla compatibilità dei kernel di Windows 8 e Windows 10 con la nuova architettura.

Appendice A

Metrica XCModel

Di seguito elenco la metrica fornita dal programma Frama-C relativamente ai moduli di XCModel. Oltre alle solite statistiche viene fornito un valore chiamato *complessità ciclomatica* (Cyclomatic complexity).
La Complessità Ciclomatica (o complessità condizionale) è una metrica software sviluppata da Thomas J. McCabe nel 1976 [CICLO], è utilizzata per misurare la complessità di un programma.
Misura direttamente il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso.
Un'applicazione della complessità ciclomatica è determinare il numero di casi di test che sono necessari per l'analisi di coverage di una particolare funziona/subroutine.

```
Descriptor:
Global metrics
          ==============
          Sloc = 1690
          Decision point = 297
          Global variables = 17
          If = 280
          Loop = 25
          Goto = 25
          Assignment = 509
          Exit point = 99
          Function = 133
          Function call = 607
          Pointer dereferencing = 674
          Cyclomatic complexity = 200
```

```
Matrix:
Global metrics
==============
          Sloc = 497
          Decision point = 56
          Global variables = 0
          If = 56
          Loop = 43
          Goto = 3
          Assignment = 240
          Exit point = 39
          Function = 40
          Function call = 20
          Pointer dereferencing = 164
          Cyclomatic complexity = 19


Trim:
Global metrics
==============
          Sloc = 10400
          Decision point = 1749
          Global variables = 175
          If = 1682
          Loop = 327
          Goto = 227
          Assignment = 4928
          Exit point = 181
          Function = 227
          Function call = 1488
          Pointer dereferencing = 4232
          Cyclomatic complexity = 1570


XCBool:
Global metrics
==============
          Sloc = 13843
          Decision point = 2472
          Global variables = 149
          If = 2354
          Loop = 422
          Goto = 535
          Assignment = 3733
          Exit point = 433
          Function = 592
          Function call = 4340
          Pointer dereferencing = 6291
          Cyclomatic complexity = 2041


XCCury:
Global metrics
==============
          Sloc = 15144
          Decision point = 2023
          Global variables = 131
          If = 1771
          Loop = 665
          Goto = 105
          Assignment = 5360
          Exit point = 452
          Function = 585
          Function call = 4217
          Pointer dereferencing = 3190
          Cyclomatic complexity = 1573
```

```
Xdbe:
Global metrics
==============
          Sloc = 2231
          Decision point = 366
          Global variables = 72
          If = 357
          Loop = 69
          Goto = 54
          Assignment = 790
          Exit point = 63
          Function = 146
          Function call = 674
          Pointer dereferencing = 597
          Cyclomatic complexity = 305


XCModel:
Global metrics
==============
          Sloc = 784
          Decision point = 63
          Global variables = 28
          If = 63
          Loop = 9
          Goto = 7
          Assignment = 138
          Exit point = 43
          Function = 96
          Function call = 479
          Pointer dereferencing = 56
          Cyclomatic complexity = 22


XCssi:
Global metrics
==============
          Sloc = 4795
          Decision point = 818
          Global variables = 83
          If = 796
          Loop = 208
          Goto = 76
          Assignment = 2029
          Exit point = 136
          Function = 194
          Function call = 836
          Pointer dereferencing = 2877
          Cyclomatic complexity = 684


XCSurf:
Global metrics
==============
          Sloc = 31278
          Decision point = 4633
          Global variables = 661
          If = 4196
          Loop = 1389
          Goto = 510
          Assignment = 12649
          Exit point = 1239
          Function = 1411
          Function call = 7340
          Pointer dereferencing = 9082
          Cyclomatic complexity = 3396
```

```
XTools:
Global metrics
==============
          Sloc = 4411
          Decision point = 786
          Global variables = 62
          If = 739
          Loop = 148
          Goto = 49
          Assignment = 1622
          Exit point = 256
          Function = 334
          Function call = 1007
          Pointer dereferencing = 2679
          Cyclomatic complexity = 532
```

Bibliografia

[MCCON] Steve McConnell. *Complete code*, Microsoft Press (1993).

[CASC01] G. Casciola, S. Morigi. *xcmodel: an aCADemic system*, Unità operativa di Bologna (anno).

[CASC02] G.Casciola. *xcsurf: the 3D modeller, User's Guide - Version 1.0*, (2000).

[CASC03] G.Casciola, G.DeMarco. *xcbool: the object composer, User's Guide - Version 1.0*, (2000).

[CASC04] G.Casciola. *xcrayt: the scene descriptor, User's Guide - Version 1.0*, (2000).

[CASC05] G.Casciola. *MATRIX library: Programming Guide - Version 1.0*, (1999).

[CASC06] G.Casciola, S.Bonetti. *descriptor library: Programming Guide - Version 1.0*, (1999).

[CASC07] G.Casciola, G.DeMarco. *trim library: Programming Guide - Version 1.0*, (1999).

[CASC08] G.Casciola, S.Bonetti. *xtools library: Programming Guide - Version 1.0*, (1999).

[CASC09] G.Casciola, S. Morigi. *xcmodel: an aCADedmic system*, Ann. Univ. Ferrara - Sez. VII - Sc.Mat., Supplemento al Vol.XLV, (2000).

[CASC10] G.Casciola, E.Trevisan. *Free Form Deformation: xcsurf (Version 2.0) plugin*, (2001).

[CASC11] G.Casciola, S.Bonetti. *xtools library: Programming Guide - Version 2.0*, (2001).

[JAEG] Andreas Jaeger. *Porting to 64-bit GNU/Linux Systems*, GCC Developers Summit (2003).

[IBM] Harsha S. Adiga. *Porting Linux applications to 64-bit systems*, IBM Developer Works, Techical Library (2006).

[KAR] Andrey Karpov. *64 bits*, Intel Developer Zone (2010).

[GAL] Matteo Galisi. XCSurf: riprogettazione dell'interfaccia utente, Tesi di Laurea, Università di Bologna (2010).

[TRIC] Adam Trickett. *Running 32-bit Applications on 64-bit Debian GNU/Linux*, www.debian-administration.org (2007)

[HP] HP. *Porting an application to 64-bit Linux on HP Integrity servers*, HP AllianceOne (2009).

[CICLO] Complessità ciclomatica. `http://it.wikipedia.org/wiki/Complessità_ciclomatica`.

Ringraziamenti

Desidero ricordare tutti coloro che mi hanno aiutato nella stesura della tesi con suggerimenti, critiche ed osservazioni: a loro va la mia gratitudine, anche se a me spetta la responsabilità per ogni errore contenuto in questa tesi.

Desidero ringraziare il professor Giulio Casciola, relatore di questa tesi, per la grande disponibilità e cortesia dimostratemi, e per tutto l'aiuto fornito durante la stesura.

Un sentito ringraziamento ai miei genitori, che, con il loro incrollabile sostegno morale ed economico, mi hanno permesso di raggiungere questo traguardo.

Vorrei infine ringraziare le persone a me più care: Giorgia, la sorellina di Giorgia, Romina, Alessio e Maria Pia.