Consiglio Nazionale delle Ricerche

Progetto HPCC-SEA
VisTool:
Strumenti di Visualizzazione in Ambiente Parallelo

R. Scopigno, E. Ciabatti

Nota Tecnica

CNUCE-B4-1999-010

CNUCE

Pisa

Progetto HPCC-SEA

VisTool

Strumenti di Visualizzazione in Ambiente Parallelo

Relazione finale

R. Scopigno, E. Ciabatti

Gruppo Visual Computing, CNUCE-I.E.I. / C.N.R.
v. S. Maria 46, 56126 Pisa ITALY
Email: roberto.scopigno@cnuce.cnr.it

Settembre 1999

Abstract

In questo documento viene descritta l'attivita' del Gruppo Visual Computing nell'ambito del Progetto *VisTool,* sviluppato in collaborazione con il gruppo *Quadrics Supercomputers World* (QSW) di Pisa. In particolare, si descrivono le scelte operate nella progettazione architetturale del sistema integrato VisTool, alcuni aspetti dell'implementazione ed infine i risultati ottenuti.

Obiettivo del progetto *VisTool* e' stato verificare l'integrabilita' di un ambiente di programmazione parallela ad alto livello (SkIE) con strumenti standard di visualizzazione (nella fattispecie, la libreria AVS). Nella realizzazione di tale ambiente integrato sono riscontrabili due principali vantaggi: da una parte viene proposto agli utilizzatori di sistemi di visualizzazione standard l'integrazione con un ambiente di programmazione parallela ad alto livello di astrazione, per sviluppare le soluzioni di task critici in termini di tempo e/o spazio; contemporaneamente, viene offerta agli sviluppatori di applicazioni in ambiente parallelo la possibilita' di usufruire di funzionalita' standard di visualizzazione, che risultano in genere necessarie come attivita' di post-processing in molteplici e differenziati ambiti applicativi.

In particolare, nella prima fase di progetto sono stati analizzati e valutati alcuni ambienti di visualizzazione; come risultato di tale analisi si e' scelto il sistema Application Visualization System - AVS. Successivamente, e' stato sviluppato il progetto architetturale dell'intero sistema e ne e' stata realizzata una implementazione.

1. Introduzione

La crescente necessità, nelle più svariate applicazioni informatiche, di fornire output in maniera grafica ha reso indispensabile la presenza di macchine e tecniche di elaborazione sempre più efficienti, sia in termini di velocità di elaborazione, sia nella capacità di visualizzare grosse quantità di dati. Computer graphics, realtà virtuale, medical imaging, data mining, simulazioni in campo chimico, fisico, meteorologico, sono solo un piccolo esempio delle aree di interesse che necessitano di pesanti elaborazioni associate a tecniche di visualizzazione rapide e precise. Applicazioni in questi campi possono richiedere, infatti, grosse quantità di dati da elaborare e tempi di risposta valutabili in ore se non addirittura giorni. La direzione da seguire è, dunque, quella di un'efficiente integrazione tra visualizzazione scientifica e calcolo ad alte prestazioni.

In quest'ambito, si pone la realizzazione di un sistema integrato per lo sviluppo di applicazioni, che consenta di sfruttare le tecnologie fino ad ora acquisite sia nel campo dell'elaborazione parallela che nell'ambito degli strumenti di visualizzazione. Tale ambiente nasce dall'integrazione di due ambienti già preesistenti: l'ambiente di visualizzazione AVS/Express e l'ambiente di programmazione parallela SkIE.

Nel seguito introdurremo brevemente le caratteristiche dei due sistemi che compongono l'ambiente integrato, AVS/Express e SkIE, e le motivazioni che hanno portato alla scelta di AVS come ambiente di visualizzazione.
Ci soffermeremo, quindi, sull'architettura del sistema integrato e sulle modalità del suo impiego.
Infine, nell'ultima sezione presenteremo, come esempio, un'applicazione parallela realizzata all'interno dell'ambiente integrato.
L'obiettivo nello sviluppo di questa applicazione e' duplice: testare l'effettiva interoperabilita' tra gli ambienti SkIE e AVS, e valutare costo implementativo, efficacia e performance di una applicazione di visualizzazione parallela sviluppata ed eseguita nell'ambiente integrato. Si tratta in particolare di un nuovo *modulo parallelo* per AVS/Express che implementa un algoritmo per l'estrazione di isosuperfici e per la loro semplificazione, in cui vengono sfruttate le potenzialità offerte dall'elaborazione parallela.

2. AVS

AVS, ed in particolare Express che ne costituisce la più recente implementazione, è un complesso sistema rivolto alla visualizzazione scientifica [UFK89]. E' utilizzato, in molti diversi settori (chimica, fisica, medicina, geologia, applicazioni di fluidodinamica...), sia da semplici utenti che da sviluppatori di applicazioni.
AVS/Express fornisce una vasta dotazione di moduli che implementano funzionalità di vario tipo, organizzate in differenti categorie logiche, da quelle più semplici di tipo input/output, a quelle più complesse tipo funzioni di filtro (conversione dei dati fra formati diversi, riduzione dei dati ecc.) o di mapping (tipo estrazione di isosuperfici ecc.).
Un generico utente, mediante l'ambiente di sviluppo visuale di AVS/Express, il Network Editor, può realizzare applicazioni complesse semplicemente istanziando in un workspace i moduli relativi alle funzioni di interesse e componendoli fra loro in una rete di calcolo di tipo *dataflow*. E' possibile anche definire nuovi moduli ed inserirli nella libreria standard. In particolare, AVS/Express si basa su una metodologia object-oriented in cui i moduli sono oggetti che includono propri dati e metodi e sono organizzati secondo un ordine gerarchico.

In definitiva, le caratteristiche che hanno portato alla scelta di AVS/Express per la definizione dell'ambiente integrato sono essenzialmente le seguenti:

- è un sistema di sviluppo *general purpose*, ovvero rivolto ad un'ampia gamma di settori di visualizzazione;
- è un sistema *aperto*, nel senso che permette non solo lo sviluppo di applicazioni stand-alone ma anche lo sviluppo di componenti da integrare nel sistema stesso;
- è portabile su un'ampia gamma di piattaforme;
- è modulare, in quanto basato su una filosofia object-oriented secondo cui le varie funzionalità sono inglobate in moduli che possono a loro volta essere connessi tra loro.

Quest'ultima caratteristica, in particolare, ne avvicina la filosofia di programmazione a quella dell'ambiente SkIE.

3. L'ambiente SkIE (*Skeleton Integrated Environment*)

Il principale ostacolo alla diffusione della tecnologia parallela è la complessità (e l'associato costo) del processo di sviluppo di software parallelo. Una buona metodologia di sviluppo per applicazioni parallele dovrebbe essere in grado di garantire programmabilità (ossia la possibilità di scrivere e modificare in modo semplice e rapido programmi paralleli e controllare la loro correttezza rispetto alle specifiche iniziali), portabilità e performance, sfruttando al massimo le caratteristiche della macchina target.
L'ambiente di sviluppo **SkIE** (***Skeleton Integrated Environment***) si prefigge di raggiungere tali obiettivi fornendo un linguaggio di coordinamento chiamato **SkIECL** (***SkIE Coordination Language***) ed un insieme di tool di supporto (interfaccia grafica, debugger, ecc.) pensati specificatamente per guidare ed

aiutare i programmatori in fase di progetto, sviluppo e testing di applicazioni parallele. L'ambiente SkIE è supportato su tutte le piattaforme QSW, su Sun Solaris, Linux e Windows NT.

Il cuore del sistema è costituito dal linguaggio SkIECL che presenta le seguenti caratteristiche fondamentali:

- è un linguaggio strutturato a parallelismo esplicito, ovvero è basato su un insieme di costrutti paralleli fondamentali componibili tra loro;
- è un linguaggio ad alto livello, ovvero permette all'utente di svincolarsi dalla gestione di aspetti critici dell'elaborazione parallela, quali comunicazioni, sincronizzazioni, bilanciamento del carico, ecc., garantendo al tempo stesso l'indipendenza dell'applicazione dalla macchina target;
- favorisce la riusabilità del codice garantendo l'uso/riuso di pezzi di codice scritti in diversi linguaggi sequenziali.

Il fatto che SkIECL sia un linguaggio parallelo strutturato significa che la struttura di un programma può essere espressa solo tramite la composizione gerarchica di un insieme di forme primitive di parallelismo (dette *costrutti paralleli*). Dunque, almeno al livello più astratto, il parallelismo viene definito esplicitamente dal programmatore.
I costrutti costituiscono, a livello di istruzioni del linguaggio, l'implementazione di un insieme di ***skeleton***, ovvero forme standard di parallelismo che sono alla base di un'ampia classe di applicazioni parallele. Componendo gli skeleton tra loro è anche possibile definire forme di parallelismo ancor più complesse.

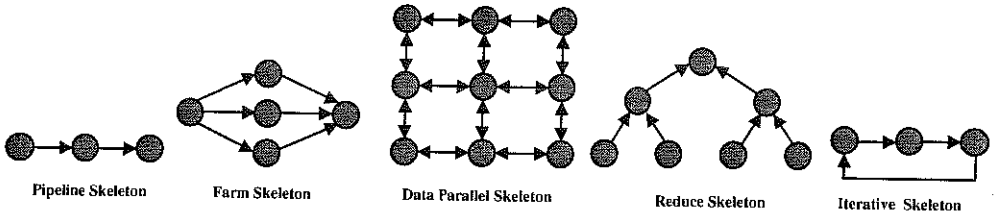


Figura 1. Skeleton di base in SkIE

La progettazione di un'applicazione parallela in SkIE consiste in definitiva nella composizione di più skeleton, alla cui base sono posti uno o più nodi (istanze di costrutti) sequenziali. In tali nodi l'utente, utilizzando uno dei possibili linguaggi host (C, C++, F77, F90, Java), può esprimere le parti di computazione prettamente non parallele.

Un altro aspetto fondamentale nell'utilizzo di SkIE è che tutti problemi implementativi dei costrutti paralleli, relativi sia alla loro strutturazione (ad esempio bilanciamento del carico o partizionamento dei dati) che alle caratteristiche dell'architettura su cui devono essere implementati (quali il numero dei nodi di elaborazione o il costo delle comunicazioni), sono a carico degli strumenti di compilazione. Il programmatore può dunque concentrarsi sulla struttura parallela dell'applicazione senza dover gestire problemi di elevata complessità dalla cui efficiente risoluzione dipenderà la performance del risultato.
Maggiori dettagli sulle funzionalità supportate dal linguaggio SkIECL e sui tool dell'ambiente SkIE sono descritte in [BCPR98].

4. Architettura del sistema integrato SkIE-AVS

Come abbiamo già accennato nell'introduzione, l'idea fondamentale che sta alla base del nostro progetto è di integrare due tecnologie consolidate in un unico ambiente di lavoro con il raggiungimento, dal punto di vista del suo utilizzo, di un duplice obiettivo:

- da una parte, si offre agli utilizzatori di un sistema di visualizzazione ampiamente diffuso la possibilità di utilizzare un ambiente di programmazione parallela ad alto livello di astrazione, per sviluppare le soluzioni di task critici in termini di tempo e/o spazio;

- dall'altra, viene offerta agli sviluppatori di applicazioni in ambiente parallelo la possibilità di usufruire delle funzionalità standard di visualizzazione fornite da AVS/Express, che risultano normalmente necessarie come attività di post-processing in vari settori di applicazione.

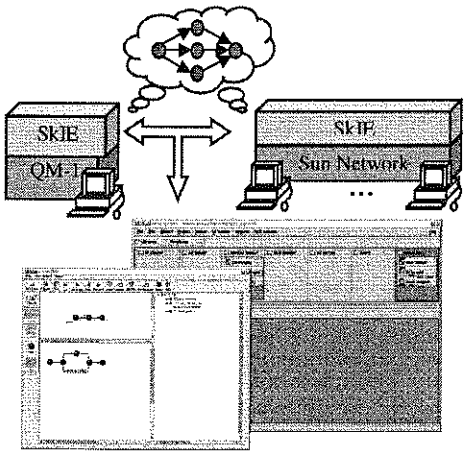


Figura 2. Architettura del sistema SkIE-AVS

Tale integrazione viene realizzata fornendo all'utente di AVS/Express la possibilità di definire dei nuovi *moduli paralleli*, ovvero moduli la cui specifica interna è scritta nel linguaggio SkIECL, e dunque eseguibili su macchine parallele, ma perfettamente integrati all'interno di AVS/Express.
Il cuore del sistema è costituito da un nuovo modulo AVS/Express chiamato *SkIE Maker*, ed inserito in una libreria *Parallel*, la cui istanziazione attiva l'ambiente di generazione dei nuovi moduli paralleli.

4.1 Architettura del sistema

L'architettura hardware dell'ambiente integrato è costituita essenzialmente da due parti principali:

- una macchina client (di tipo Sun/Solaris) su cui viene effettuata l'esecuzione del sistema AVS/Express e dell'interfaccia grafica dell'ambiente di programmazione parallela SkIE (VisualSkIE);
- uno o più server (macchine Quadrics QM-1 o più in generale reti di tipo Sun/Solaris o Linux) che costituiscono la piattaforma parallela per l'esecuzione parallela dei nuovi moduli definiti nell'ambiente integrato.

Ovviamente è possibile che la macchina host sia collegata a più di una macchina parallela. L'unica restrizione per far si' che una macchina possa far parte del sistema integrato, è che la home dell'utente sia esportata sulla macchina client tramite Network File System.
Dal punto di vista software l'ambiente integrato si compone di tre elementi principali (Figura 2):

- il sistema AVS/Express;
- l'ambiente di programmazione parallela SkIE, composto di:
  - interfaccia grafica Visual SkIE (e ambiente Java su cui è stata sviluppata);
  - compilatore del linguaggio SkIECL (su macchina astratta MPI che ne costituisce il target e almeno un ambiente di sviluppo sequenziale (C, C++, FORTRAN);
- il modulo di libreria, ***SkIE Maker*** che realizza l'integrazione fra il tool di visualizzazione e l'ambiente parallelo.

4.2 Il modulo SkIE Maker

Il cuore del sistema integrato è il modulo *SkIE Maker* che implementa fondamentalmente due funzionalità principali:

- interazione di AVS/Express con l'ambiente di programmazione parallela, e quindi, dal punto di vista della comunicazione diretta, con l'interfaccia Visual SkIE;
- integrazione nell'ambiente AVS/Express di un generico programma parallelo, sviluppato tramite SkIE; ovvero costruzione di un nuovo modulo Express che ne inglobi le funzionalità del programma parallelo.

Nel primo caso si tratta di realizzare una sorta di comunicazione per lo scambio di informazioni tra i due ambienti in modo che le informazioni relative alla configurazione dell'ambiente parallelo e alle specifiche del programma definito dall'utente siano rese accessibili nell'ambiente Express.
Nel secondo si tratta di integrare tutte le informazioni raccolte per costruire un nuovo *modulo parallelo* che risponda alle esigenze di implementazione di entrambi i sistemi.
Non ci soffermeremo oltre sugli aspetti di carattere implementativo del sistema ma, preferendo il punto di vista dell'utente, forniremo invece la descrizione di una tipica sessione di lavoro a partire dalla istanziazione del modulo *SkIE Maker*.

Inizializzazione della sessione di lavoro in ambiente *VisualSkIE*
Il primo passo consiste nella semplice visualizzazione, per il tempo necessario al caricamento del sistema, di una finestra di presentazione che introduce l'utente nella sessione di lavoro del sistema integrato. Al termine del caricamento viene presentata la finestra relativa all'interfaccia del linguaggio SkIECL (passo 2) e il controllo passa all'ambiente *VisualSkIE*. Allo stesso modo di Express, anche VisualSkIE (Figura 3) viene eseguito sul client e dunque la fase di sviluppo del programma parallelo da parte dell'utente avviene localmente. L'utente può interrompere la sessione in qualunque momento (passo 3), indipendentemente dallo stadio di sviluppo del suo progetto, salvando eventualmente il progetto in corso per poi richiamarlo in una sessione successiva. All'uscita dall'ambiente VisualSkIE, se non è stato generato alcun eseguibile non verranno eseguite azioni ulteriori e il controllo tornerà al Network Editor di Express.

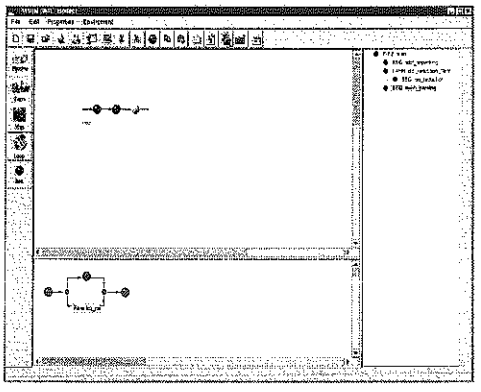


Figura 3: Esempio di programma in ambiente VisualSkIE

Definizione delle specifiche del modulo parallelo
Affinché Express possa decidere (passo 3) se effettuare o meno gli ulteriori passi di integrazione è necessario un passaggio di informazioni tra i due sistemi. L'informazione fondamentale è relativa all'esistenza o meno di un programma parallelo eseguibile e, in caso affermativo, ingloba anche informazioni che caratterizzano il programma (nome, collocazione, numero e nomi dei suoi eventuali parametri di ingresso).
La prosecuzione della sessione di lavoro, consiste (passo 4) nella specifica da parte dell'utente, mediante una apposita finestra di interfaccia (Figura 4), delle caratteristiche che dovrà avere il nuovo modulo *parallelo*. Tali specifiche sono in parte dipendenti dal programma da integrare (numero e tipo dei

parametri di ingresso) e dunque automaticamente definibili dal sistema; in parte possono dipendere dall'utilizzo che verrà fatto del modulo e, dunque, devono essere specificate dall'utente. Ad esempio, un modulo parallelo potrà disporre di un numero di porte di ingresso nullo o pari al più al numero di parametri di ingresso del programma (cui le porte sono associate). In particolare, per ciascun parametro è possibile specificare

- se dovrà essere una porta di ingresso (*display as input port*),
- se una variazione del suo valore dovrà determinare l'esecuzione del modulo (*notify*),
- se il parametro dovrà essere visibile all'interno del modulo (*export parameter*).

Per quel che riguarda i parametri di uscita è stabilito staticamente che un qualunque modulo parallelo abbia un'unica porta di uscita (*Ok*) il cui valore, istanziato al momento della terminazione del modulo, definisce l'esito dell'esecuzione stessa.
Terminata la specifica della tipologia dei parametri, il sistema deve generare automaticamente il codice del modulo sia per la parte di computazione vera e propria, descritta in linguaggio C, sia per la descrizione V necessaria alla sua visualizzazione all'interno di Express. Deve inoltre stabilire una collocazione visuale del modulo all'interno di Express e le sue relazioni con gli elementi preesistenti.

Dunque nella stessa finestra in cui sono definiti i parametri, è necessario che l'utente specifichi anche ulteriori informazioni quali:

- il nome del nuovo modulo che sarà generato, nel campo *'Module name'*;
- i nomi dei file di definizione C e V, rispettivamente in *'Source Filename' e 'File V'*;
- la locazione di Express, intesa come libreria di moduli, in cui il modulo dovrà essere caricato (per default *Workspace_1*), nel campo *'To load in'*;
- il nome del processo cui il modulo dovrà, eventualmente, essere associato (per default *user*), nel campo *'Process name'*.

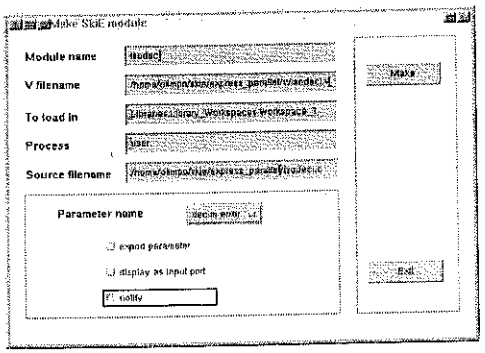


Figura 4: Finestra di specifica del nuovo modulo

Generazione e compilazione del modulo
Fornite tutte le informazioni necessarie, l'utente può a questo punto richiedere la generazione del nuovo modulo (passo 5). Quindi, nell'ipotesi che questa sia andata a buon fine può decidere se effettuare o meno la fase, opzionale, di integrazione e compilazione del modulo all'interno di un processo Express (che non sia ovviamente quello attualmente in esecuzione) già esistente. Qualunque modulo in Express, infatti, deve essere integrato all'interno di un progetto ma l'utente potrebbe voler decidere di effettuare manualmente le necessarie modifiche dei file di descrizione del progetto. Una finestra di dialogo (passo 6), dunque, avvisa l'utente della corretta generazione del modulo e richiede se effettuare o meno la sua effettiva integrazione (passo 7). Indipendentemente dalla scelta effettuata, comunque, il risultato finale della sessione di lavoro consiste nell'inserimento di un nuovo modulo in una specificata libreria di Express.

5. Un esempio di applicazione

Un modulo *parallelo* generato mediante il sistema integrato è, come abbiamo ripetuto più volte, a tutti gli effetti un modulo Express. Dunque può essere utilizzato all'interno di applicazioni più complesse insieme con altri moduli standard o generati dall'utente.
Al fine di verificare le potenzialità d'uso e quantificare la validità dei risultati ottenibili con l'utilizzo dell'ambiente integrato, abbiamo sviluppato una applicazione di visualizzazione il cui cuore e' un nuovo modulo, *isodeci*, che esegue, in parallelo l'estrazione di isosuperfici da dataset tridimensionali e la loro semplificazione. Nei paragrafi successivi ci soffermeremo sulle motivazioni che hanno portato alla scelta di parallelizzare questo particolare algoritmo e sulle scelte architetturali operate.
Ora ci limitiamo ad osservare (Figura 5) come il modulo *isodeci* può essere utilizzato per costruire una generica applicazione in connessione sia con filtri definiti dall'utente (*fld_to_sdsl, mesh_mapper*), sia con funzionalità fornite da AVS/Express (*tri_mesh, Uviewer3D*).
In particolare, utilizzando la rete raffigurata, è possibile, tramite i filtri, utilizzare all'interno di Express un algoritmo (*isodeci*) definito per dati in formato *sdsl*, e dunque diverso dal field standard di AVS. Inoltre è evidente come funzionalità preesistenti, quali in questo caso il visualizzatore, vengono facilmente sfruttate per effettuare una attività di post-processing necessaria alla valutazione dei risultati dell'algoritmo implementato.

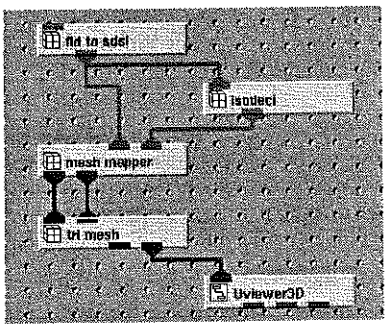


Figura 5: Esempio di applicazione

5.1 Visualizzazione di dati volumetrici

La tipologia di dati che ci interessa prendere in considerazione è costituita dai dati volumetrici (o *voxel* dataset). Come esempio possiamo considerare ai dati prodotti da apparecchiature di tomografia assiale computerizzata (TAC), che generano nuvole di dati distribuiti regolarmente in un volume rettilineo. Dataset reali sono in genere organizzati per da slice (griglie bidimensionali), e la loro complessita' spaziale è direttamente dipendente dalla risoluzione di acquisizione (ad es. un dataset 512x512x80 richiede circa 40 MB). Dataset volumetrici possono anche essere prodotti da simulazioni al computer relative allo studio di fenomeni fisici o chimici.

Lo studio e l'analisi di tali dataset è oggi essenzialmente di tipo visivo (ed interattivo). Una diffusa strategia consiste nell'evidenziare una serie di valori del campo rappresentato che rispondono a determinati requisiti e visualizzare la distribuzione spaziale di tali valori nel dataset generando delle cosiddette *isosuperfici*. Ovviamente, variando il valore del campo specificato si ottengono isosuperficie diverse. Alcune isosuperfici sono rappresentate in Figura 11.

Per l'*estrazione* di una isosuperfice da un dataset volumetrici si fa in genere uso dell'algoritmo *Marching Cubes* [LC87,MSS94b], che produce in uscita una rappresentazione della superficie costituita da una mesh di triangoli.
La complessità delle isosuperfici estratte dipende sia dalla risoluzione del dataset che dalla distribuzione nel dataset del particolare valore del campo selezionato; su dataset sufficientemente complessi la generazione di isosuperfici costituite da centinaia di migliaia o milioni di facce è quindi abbastanza frequente.

La complessità in spazio sia dell'originale dataset volumetrico che delle isosuperfici estratte (si può arrivare facilmente a diverse centinaia di megabyte) rende difficoltosa la gestione interattiva dell'estrazione e visualizzazione di isosuperfici anche su architetture di alto livello.
Alcune tecniche per la *semplificazione* di rappresentazioni basate su superfici poligonali (ossia per la riduzione controllata della complessità, misurata in genere come numero di facce costituenti) sono state recentemente sviluppate [PS97, SZL92,CCMS97, GH97].Tali tecniche permettono di ridurre in modo sostanziale complessità delle isosuperfici estratte (e, conseguentemente, I tempi di visualizzazione), ma richiedono un tempo di elaborazione e occupazione di memoria (a tempo di esecuzione) direttamente dipendente dalla complessità dei dati. Ciò ne rende difficile l'uso nel caso di dataset estremamente complessi.

5.2 Surface-based volume visualization in ambiente parallelo

Per rendere l'intero processo più efficiente e veloce sono state parallelizzate le fasi di estrazione e di semplificazione di isosuperfici, utilizzando il linguaggio SkIE [BCPR98]. I problemi sopracitati sono stati affrontati adottando una approccio basato sul partizionamento del dataset, in modo di suddividerlo in sotto-blocchi più piccoli e più facilmente gestibili. L'isosuperficie estratta dai singoli blocchi viene quindi semplificata, in modo da ridurne la complessità pur preservandone l'aspetto in fase di visualizzazione. I risultati ottenuti sui singoli blocchi vengono fusi insieme in una fase di post-processing ottenendo la superficie finale.

I vantaggi del nostro approccio si possono riassumere nei seguenti punti:

- permette di gestire dataset di grande dimensione, con l'unico limite della dimensionie della memoria secondaria;
- si incrementa la velocità di estrazione, grazie alla parallelizzazione;
- permette di visualizzare in modo più efficiente l'isosuperficie estratta, grazie alla semplificazione della geometria.

Dai diagrammi presentati nelle Figure 6 e 7 si ricava la struttura generale dell'applicazione.
L'ambiente di visualizzazione AVS Express si prevede sua eseguito sulla macchina client sequenziale. Per scambiare i dati tra la macchina sequenziale e quella parallela viene utilizzato il Network File System (NFS). Il dataset volumetrico si prevede sia memorizzato in file su disco (in formato *sdsl*, vedi Appendice A). L'applicazione sulla macchina parallela legge il file dei dati, estrae una isosuperficie, la semplifica ed infine salva il risultato (una mesh triangolata) in un file di output. A questo punto AVS potra' leggere il risultato e visualizzarlo attraverso uno dei moduli standard di visualizzazione 3D.

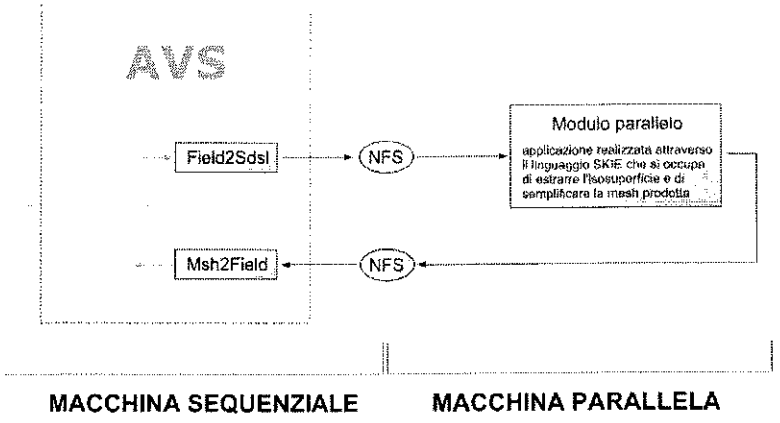


Figura 6. Interazione tra modulo parallelo IsoDeci ed ambiente di visualizzazione AVS.

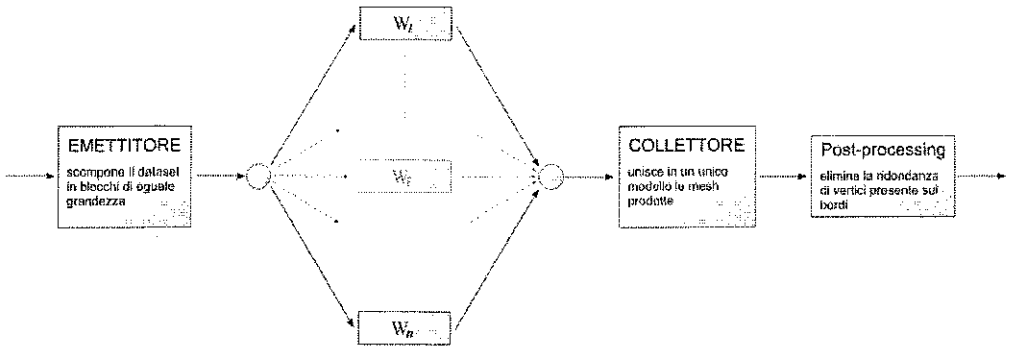


Figura 7. Architettura del modulo parallelo IsoDeci.

La struttura generale segue il modello *pipeline-farm*, ed in fase di compilazione verra' generato un processo per ogni blocco del diagramma.
Il processo emettitore legge il dataset in input e lo scompone in blocchi di eguale grandezza (specificata dall'utente) inviandoli al nodo successivo che è una *farm*. La farm è costituita da *n* worker il cui numero è stabilito a tempo di compilazione in base al numero di nodi disponibili della macchina parallela ed alla mole di dati da elaborare. Tutti i worker svolgono la stessa funzione costituita dall'estrazione dell'isosuperficie e dalla sua semplificazione. L'output dei worker viene inviato ad un processo collettore che procede ad assemblare la superficie finale. Infine una fase di post-processing provvede ad eliminare la ridondanza di vertici generata dalla suddivisione scrivendo in un file il modello finale dell'isosuperficie.

Il modulo parallelo viene invocato attraverso un comando remoto (rshell) e prende in input alcuni parametri a linea di comando: il filename del dataset da elaborare; la dimensione del sotto-blocco in x, y e z; il valore soglia utilizzato per l'estrazione dell'isosuperficie; l'errore in voxel utilizzato per la semplificazione dell'isosuperficie.

Nei paragrafi seguenti sono descritte in dettaglio le singole fasi dell'applicazione.

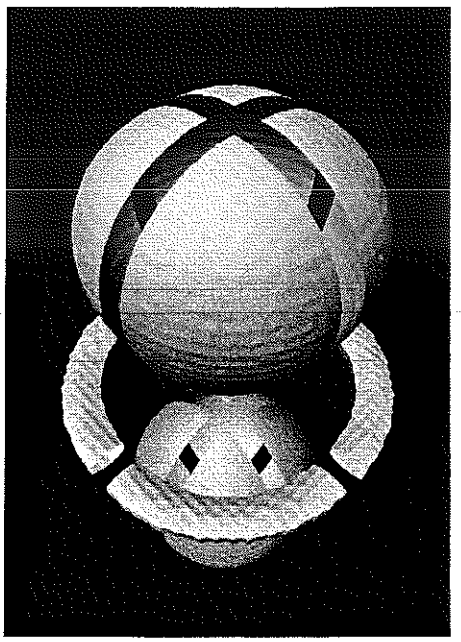


Figura 8. Una isosuperficie estratta dal dataset *Hydro* suddiviso in 8 blocchi.

5.2.1 Emettitore

Il processo emettitore si occupa di suddividere il dataset in ingresso in sotto-blocchi di eguale grandezza. Il programma produce una serie di file ognuno dei quali memorizza il contenuto di una parte del volume originario.
L'utente, attraverso i parametri a linea di comando, specifica la dimensione dei blocchi che devono essere prodotti. Se la dimensione del volume non è un multiplo esatto del valore specificato dall'utente allora il programma produrrà dei blocchi di dimensioni inferiori in corrispondenza dei bordi del volume.
Il programma realizza due ottimizzazioni:

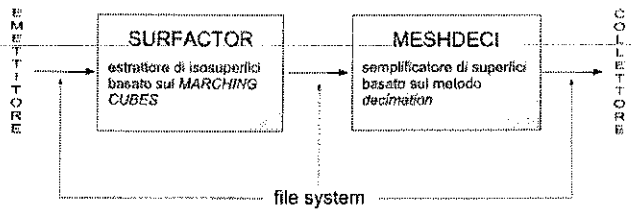
- il volume dei dati è memorizzato in slice elencate lungo l'asse zeta. Dunque è possibile leggere il file fino a raggiungere la dimensione zeta impostata dall'utente e produrre subito i blocchi suddividendo il volume caricato in memoria lungo gli assi x e y. In questo modo il blocco farm può procedere ad elaborare i dati immediatamente mentre avviene la suddivisione. Attraverso questa tecnica nell'elaborazione di dataset di grandi dimensioni si è limitati solamente dalla dimensione della memoria secondaria;

- il processo di suddivisione in blocchi viene effettuato solo in corrispondenza dell'estrazione della prima isosuperficie; se l'utente esegue successivamente l'applicazione con nuovi parametri sullo stesso dataset, il processo nel caso non sia variata la dimensione del blocco riutilizza la suddivisione operata in precedenza. Ciò consente di non replicare inutilmente la fase di suddivisione del dataset.

5.2.2 Farm e worker

Il costrutto *farm* offerto dal linguaggio SkIE consente di definire un processo *worker*, usando un qualsiasi linguaggio di programmazione (C, C++ o FORTRAN), e di replicarlo automaticamente su più nodi elaborativi. Il numero dei *worker* deve essere stabilito in modo statico al momento della compilazione in base all'architettura disponibile ed alla mole di lavoro attesa.
A basso livello, l'implementazione del costrutto *farm* istanzia un *emitter*, dei *worker* ed un *collector* che cooperano tra loro per realizzare il parallelismo: l'emitter riceve in input un flusso di dati e li distribuisce, bilanciando il carico, tra i worker disponibili; il collettore si occupa di raccogliere gli output e renderli disponibili come un flusso di dati. L'utente non deve preoccuparsi dell'interazione fra i vari worker, ma deve solamente fornire un programma che elabori i dati in ingresso (in questo caso un blocco di dati) come se fosse un semplice costrutto sequenziale.
E' bene sottolineare la distinzione tra il processo emettitore/collettore dell'applicazione (definito dall'utente) e l'emettitore/collettore nascosto nell'implementazione del farm (introdotto dal compilatore del linguaggio SkIE).

I worker relativi all'applicazione IsoDeci, sono costituiti dai due programmi *Surfactor* e *MeshDeci*.



Surfactor è un programma che realizza l'estrazione di isosuperfici da un volume di dati tridimensionale. Fra i parametri che è possibile impostare, vi è il valore di soglia per cui andrà calcolata l'isosuperficie. Il metodo utilizzato dall'algoritmo è il Marching Cubes [LC87].

MeshDeci effettua una semplificazione della superficie in input attraverso il metodo "decimation" [SZL92,CCMS97], permettendo di ridurne la complessità geometrica senza alterarne la topologia.

L'algoritmo consiste in un processo iterativo nel quale ogni vertice viene prima classificato e successivamente eliminato se l'errore globale introdotto è minore di un valore fornito dall'utente. Il processo iterativo termina quando non è possibile togliere alcun vertice dalla mesh senza superare il livello d'errore impostato dall'utente. L'efficienza del metodo è proporzionale alla complessità del modello: in mesh molto grandi (ad esempio dell'ordine del milione di triangoli) il metodo produce ottimi risultati [CCMS97]; un esempio e' riportato in Figura 9.

Questi due programmi sono stati sviluppati dal Visual Computing Group (I.E.I.-C.N.R.) ed erano disponibili come applicazioni dotate di una interfaccia utente grafica (GUI).

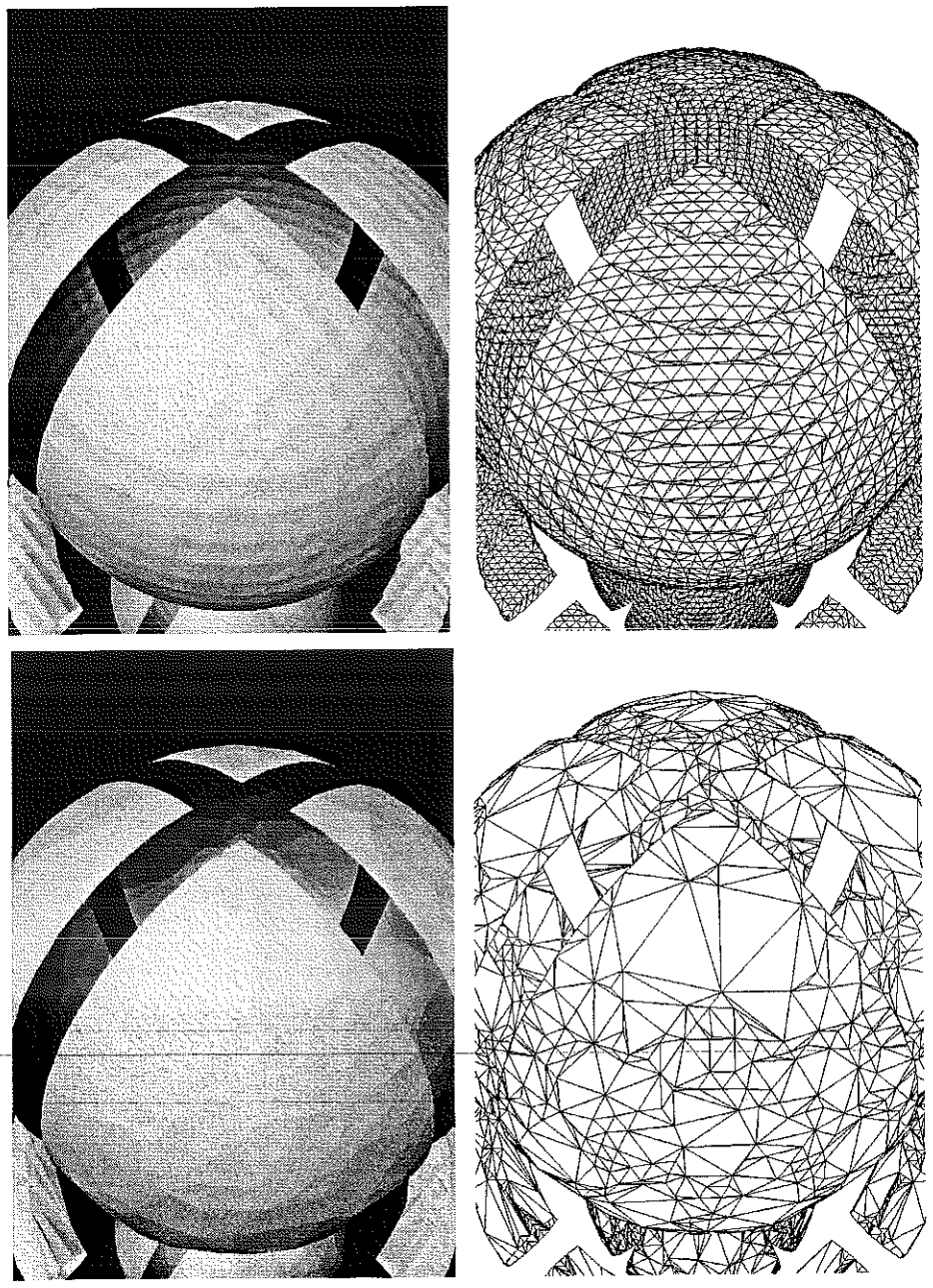


Figura 9. Semplificazione dell'isosuperficie estratta dal dataset hydro128; il coefficiente d'errore è impostato a 0.5 voxel.

Per realizzare l'applicazione *IsoDeci* è stato necessario effettuare una serie di modifiche ai codici suddetti, ed in particolare:

- il porting del codice dall'ambiente operativo originale (Irix Silicon Graphics) all'ambiente operativo di progetto (Unix Sun);

- la trasformazione del codice, originariamente costituito da due applicazioni dotate di interfaccia grafica interattiva, in una libreria di funzioni, con parsing dei parametri a linea di comando in modo da poterne eseguire le funzioni in un ambiente tipo shell Unix.

I worker invocano i due programmi attraverso una chiamata alla funzione *system()* del linguaggio C, con gli opportuni parametri (il path del blocco, la soglia di estrazione o l'errore di decimazione). Il risultato prodotto dal worker e' un file in formato *msh* che contiene la superficie semplificata relativa al blocco di dati elaborato.

L'applicazione *IsoDeci* utilizza il programma di *MeshDeci* per semplificare le isosuperfici estratte dai vari blocchi tenendo in considerazione due aspetti:

- l'utente specifica un errore globale, indicato in frazioni di voxel, che viene applicato ad ogni istanza del semplificatore *MeshDeci*. In questo modo si ha la garanzia che mesh derivanti da blocchi diversi vengano semplificate dello stesso fattore in modo da avere alla fine un modello ricomposto semplificato in maniera uniforme;

- allo scopo di unire le diverse mesh prodotte è necessario non effettuare semplificazione sui bordi. Questo vincolo porta ad avere lungo le linee di suddivisione una concentrazione maggiore di triangoli. L'effetto è direttamente proporzionale al fattore di semplificazione specificato dall'utente: partendo da grosse mesh e fornendo un coefficiente di errore elevato (2 o 3 voxel) si noterà maggiormente il problema.

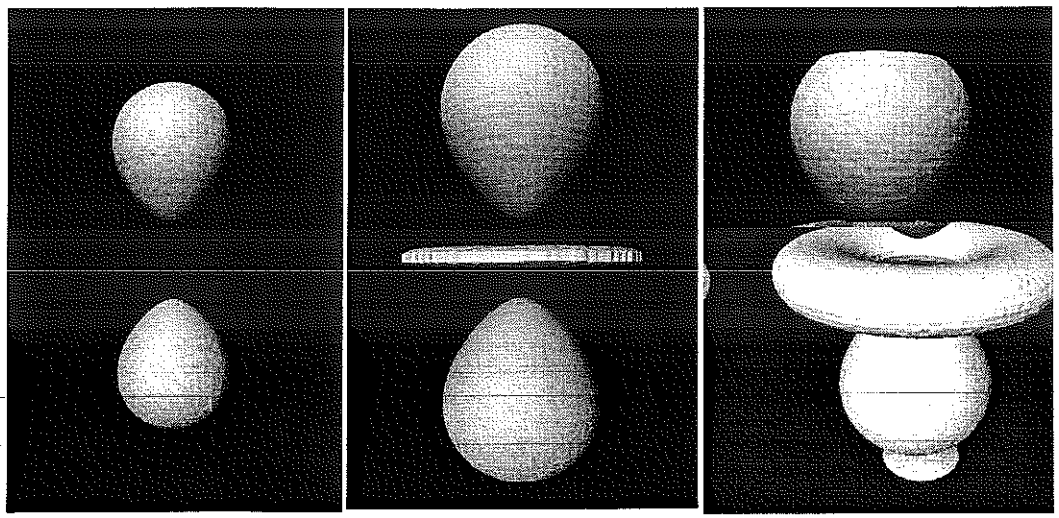


Figura 10. Estrazione di una isosuperficie dal dataset "hydro128" a differenti valori di soglia, rispettivamente 180, 127 e 80.

5.2.3 Collettore

Il collettore si occupa di riunire le mesh, estratte e successivamente semplificate dai worker, in un modello unico. Le mesh prodotte utilizzano il formato RAW, cioè' sono costituite da una semplice lista di triangoli descritti attraverso i loro tre vertici. L'operazione di unione si può effettuare semplicemente aggiungendo (*append*) i triangoli di una alla fine della lista dell'altra.
La ridondanza dei dati (vertici geometricamente uguali, che pero' in quanto presenti su più sezioni di mesh sono rappresentati da indici diversi) può essere eliminata in una fase di post-processing, in cui si ricrea la connessione fra i vari e si trasforma la mesh in formato INDEXED.

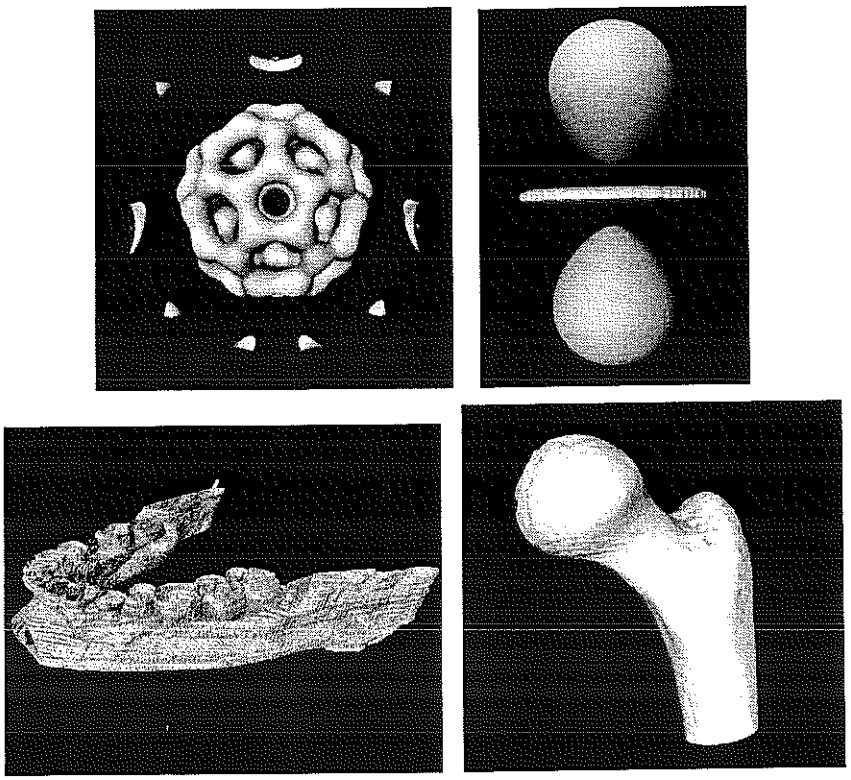


Figura 10. Isosuperfici estratte dai quattro dataset: *Bucky*, *Hydro*, *Teeth* e *Femur*.

6. Risultati

Le prestazioni dell'applicazione proposta sono state valutate su alcuni dataset:

- **Hydro**, size 127x127x127, circa 4MB;
- **Bucky**, size 128x128x128, circa 4MB;
- **Teeth**, size 523x455x40, circa 18MB;
- **Femur**, size 512x512x453, circa 226MB;

I risultati ottenuti sono riportati nelle seguenti tabelle.

Per quanto riguarda il dataset *Hydro* e *Bucky* (dataset di origine chimica) si è testata l'applicazione con 8 e con 4 worker. Come è possibile notare, 4 worker ottimizzano l'utilizzo dell'architettura parallela come si evince dai fattori di speed-up ed in particolare dalle percentuali di utilizzo riportate a fianco.
Con dataset più grandi, in questo caso di origine medicale (*Teeth* e *Femur*), 8 worker si sono rivelati più efficienti (qui sono riportate solo le tabelle con 8 worker).

L'applicazione ha prodotto buoni risultati: con 8 worker si è ottenuto uno speed-up medio di 6.2 rispetto ad una analoga versione sequenziale. Il processo di decimazione è preponderante rispetto all'estrazione ed agli altri algoritmi, come è possibile vedere nelle tabelle relative ai dataset *Hydro* e *Bucky*. Con un coefficiente di errore estremamente contenuto (0.1 voxel) si sono ottenute mesh semplificate costituite da circa 1/3 dei triangoli originali.

Blocks	Size	Seq. Time	Par. time	Speed-up	No decr. time	Triangle
8	64x64x64	12:55	2:17	5.65 70.0%	1:03	30968
16	64x64x33	12:58	2:50	4.57 57.1%	1:34	31194
27	44x44x44	13:40	2:45	4.96 62.0%	1:33	30730
54	44x44x23	13:48	2:39	5.20 65.0%	1:47	31542

Tab. 1: dataset Hydro -- 8 worker, MC, errore 0.1 voxel, soglia 80, triangoli estratti 83580

Blocks	Size	Seq. Time	Par. time	Speed-up
8	64x64x64	12:55	3:30	3.69 92.2%
16	64x64x33	13:00	3:21	3.88 97.0%
27	44x44x44	13:40	4:16	3.20 80.0%
54	44x44x23	13:48	4:30	3.06 76.0%

Tab. 2: dataset Hydro --- 4 worker, MC, errore 0.1 voxel, soglia 80, triangoli estratti 83580

Blocks	Size	Seq. time	Par. time	Speed-up	No decr. time	Triangle
8	65x65x65	20:37	3:13	6.4 80.0%	1:34	55532
16	65x65x33	20:18	3:30	5.8 72.5%	2:24	56992
27	44x44x44	21:33	3:50	5.62 70.2%	2:03	57684
54	44x44x23	22:28	3:30	6.41 80.1%	2:27	59598

Tab. 3: dataset Bucky -- 8 worker, errore 0.1 voxel, soglia 127, triangoli estratti 183480

Blocks	Size	Seq. time	Par. time	Speed-up
8	65x65x65	20:37	7:21	2.80 70.0%
16	65x65x33	20:18	5:08	3.95 98.7%
27	44x44x44	21:33	6:21	3.39 84.7%
54	44x44x23	22:28	6:17	3.57 89.4%

Tab. 4: dataset Bucky -- 4 worker, errore 0.1 voxel, soglia 127, triangoli estratti 183480

Blocks	Size	Error	Seq. time	Par. time	Speed-up	Triangle
25	106x92x40	0.01	30:05	4:57	6.06 75.0%	212712
25	106x92x40	0.1	31:28	5:17	5.95 74.0%	129855
25	106x92x40	1	33:25	5:33	6.01 75.0%	55343

Tab. 5: dataset Teeth -- 8 worker, MC, soglia 250, triangoli estratti 350468

Blocks	Size	Error	Seq. time	Par. time	Speed-up	Triangle
64	128x128x114	0.1	47:34	9:08	5.2 65.0%	203871

Tab. 6: dataset Femur -- 8 worker, soglia 127, triangoli estratti ~529K

Il dataset *Femur* inoltre dimostra che l'applicazione è in grado di gestire dataset estremamente onerosi in termini sia di spazio che di tempo di elaborazione/visualizzazione.

Più in generale, lo sviluppo dell'applicazione descritta ha permesso di verificare empiricamente l'effettivo costo di parallelizzazione di una applicazione sequenziale. Nel caso dell'applicazione testata si è verificato come un programmatore assolutamente non esperto dell'ambiente SkIE sia stato in grado, in tempi sufficientemente brevi (circa 6mesi), di utilizzare il linguaggio SkIECL per ristrutturare l'applicazione esplicitandone il parallelismo.
Questo ci ha dimostrato come l'integrazione tra diverse tecnologie, e il conseguente riuso delle conoscenze in vari campi di ricerca, può determinare dei positivi miglioramenti in tutte le aree coinvolte, dal punto di vista sia di un ampliamento degli strumenti di sviluppo sia di un netto miglioramento dei risultati ottenibili.

Bibliografia

[BCPR98] B. Bacci, B. Cantalupo, P. Pesciullesi, R. Ravazzolo, A. Riaudo, M. Torquati, L. Vanneschi, "SkIE: User Manual V2.0", 1998.

[BCCDS99] B. Bacci, B. Cantalupo, E. Ciabatti, C. Di Sacco, R. Scopigno,"Integrazione ed estensione di AVS Express in ambiente parallelo", II M.A.U.G., Montelibretti (RM), 15-17 settembre 1998 (in press).

[BDOPV95] B. Bacci, M. Danelutto, S. Orlando, S. Pelagatti, M. Vanneschi, "P3L: A structured high level programming language and its structured support,.in Concurrency: Practice and Experience", vol. 7 n. 3, Maggio 1995, pagine 225-255.

[CCMS97] A. Ciampalini, P. Cignoni, C. Montani, R. Scopigno, "Multiresolution Decimation based on Global Error", The Visual Computer, vol. 13, Giugno 1997, pp. 228-246.

[GH97] M. Garland, P.S. Heckbert, "Surface simplification using quadric error metrics", Comp. Graph. Proc., Annual Conf. Series (Siggraph '97), 1997, pp.209--216.

[LC87] W. E. Lorensen, H. E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Computer Graphics (SIGGRAPH '87 Proceedings), vol. 21, Luglio 1987, pp. 163-170.

[MSS94] C. Montani, R. Scateni, R. Scopigno, "Discretized Marching Cubes", Visualization '94 Proceedings, IEEE Computer Society Press, 1994, pp. 281-287.

[MSS94b] C. Montani, R. Scateni, R. Scopigno, "A modified look-up table for implicit disambiguation of Marching Cubes", The Visual Computer, Springer International, vol. 6, 1994, pagine 353-355.

[MSS98] C. Montani, R. Scateni, R. Scopigno, "Decreasing Iso-Surface Complexity via Discrete Fitting", Computer Aided Geometric Design, Elsevier Science, 1998 (in stampa).

[Pel97] S. Pelagatti, "Structured development of parallel programs", Taylor&Francis 1997.

[SZL92] W. J. Schroeder, J. A. Zarge, W. E. Lorensen, "Decimation of triangle meshes", ACM Computer Graphics (SIGGRAPH '92 Proceedings), vol. 26, Luglio 1992, pagine 65-70.

[UFK89] Upson, C., T. Faulhaber, D. Kamins, et al., "The Application Visualization System: A Computational Environment for Scientific Visualization", IEEE Computer Graphics & Application, Luglio 1989, pagine 30-42.

Appendice A

Il formato "sdsl"

Il formato sdsl è molto semplice e descrive un volume tridimensionale (un parallelepipedo) di valori distribuiti uniformemente nello spazio. I valori sono a 16 bit (2 byte).

Il formato sdsl è codificato in binario.

Nel file la **x** cicla più rapidamente, segue la **y** ed infine la **z**. Dunque il file è organizzato come una lista di slice parallele agli assi x-y e giacenti lungo l'asse z:

```
<x1-y1-z1>.......<xn-y1-z1>        |
                                   |
            ...                    |   prima slice x-y
                                   |
<x1-ym-z1>.......<xn-ym-z1>        |

             .
             .
             .

<x1-y1-zw>.......<xn-y1-zw>        |
                                   |
            ...                    |   ultima slice x-y
                                   |
<x1-ym-zw>.......<xn-ym-zw>        |
```

Oltre al file .sdsl vi è un altro file (.sdh) che costituisce l'header descrittivo del volume del volume di dati. Il formato è ascii:

`dimx dimy dimz`	dimensione del dataset in voxel
`voxelx voxely voxelz`	dimensione del voxel (se omogeneo è 1.0 1.0 1.0)
`shiftx shifty shifz`	spostamento del dataset nello spazio

Il formato "mesh"

Il formato "mesh" è utilizzato per descrivere una superficie triangolata. Ne esistono due versioni binarie: RAW e INDEXED.

Il formato binario RAW è una lista di triangoli non ordinata ottenuta elencando le componenti dei vertici dei triangoli:

```
<float v1.x><float v1.y><float v1.z>     |
<float v2.x><float v2.y><float v2.z>     |   primo triangolo (v1, v2, v3)
<float v3.x><float v3.y><float v3.z>     |

              .....
```

```
                .....

<float v(n+0).x><float v(n+0).y><float v(n+0).z>          |
<float v(n+1).x><float v(n+1).y><float v(n+1).z>          |   n-esimo triangolo
<float v(n+2).x><float v(n+2).y><float v(n+2).z>          |
```

Il formato INDEXED ricalca il formato canonico utilizzato da OpenGL ed è una mesh indicizzata, cioè si specifica una lista di vertici (nella quale ogni vertice è unico) e successivamente una lista di triangoli che vi si riferiscono tramite indici:

```
INDEXED                                  tag identificativo del formato (7 char)

<int>                                    numero di vertici (4 byte)
<float x><float y><float z>                   primo vertice (12 byte in tutto)
                              ....
<float x><float y><float z>                   ultimo vertice

<int>                                    numero dei triangoli (4 byte)
 <int><int><int>                              primo triangolo (12 byte)
                ....
 <int><int><int>                              ultimo triangolo
```

Il formato INDEXED produce file molto più compatti (la metà dello spazio occupato dal formato RAW) poiché si evita la ridondanza relativa ai vertici.

Appendice B

Listati (parte dei...)

Il programma ***Isodeci.sk*** definisce la struttura generale dell'applicazione parallela e comprende il codice completo del processo ***emitter***. I **worker** contengono solamente le chiamate ***system()*** dei due programmi **Surfactor** e **MeshDeci** (che si e' scelto di non riportare in appendice per brevita', ma sono comunque disponibili in sorgente).

```
/*

Estrazione di isosuperfici e decimazione delle mesh prodotte.

Versione parallela in SkIE.

Programmazione: Enrico Ciabatti

*/

#define MAX_PATH 200
#define MAX_EXT  20



/*********  E M I T T E R  *********/


sdsl_unpackingin(
                    char dataset[MAX_PATH],
                    int sizex, int sizey, int sizez,
                    float threshold,
                    float globalerrorin
                )

                out(stream of {
                    char basedataset[MAX_PATH],
                    char extension[MAX_EXT],
                    float threshold_out,
                    float globalerror,
                    int skie_error
                })

$c{

     struct VolumeData data;

     int slicex, slicey, slicez;       /* Il volume e' organizzato elencando le */
                                       /* componenti nell'ordine z, y e x (la x gira piu'
veloce) */

     short int *volume;
     short int wordbuf;
     int lenslicez, tot, n, sx, sy, sz;
     int oldsizex, oldsizey, oldsizez;
     int wx, wy, wz;
     char filename[256], *ptr;
     FILE *input, *output;
     float diagonal;
     int blockid=0;



     /* stampa il tipo di architettura */

     PrintArchitecture();
     putchar('\n');
```

```
/* toglie l'estensione */

strcpy(basedataset, dataset);
ptr=strrchr(basedataset,'.');

if (ptr)
{
     if (ptr>strrchr(basedataset,'/') && ptr>strrchr(basedataset,'\\'))
          *ptr='\0';
}


/* si rimuove la mesh finale in modo da non aggiungervi ulteriori mesh */

sprintf(filename, "%s_f.msh", basedataset);
remove(filename);



/* si settano i valori di uscita. */

skie_error=1;          /* per default si attiva l'errore  */
extension[0]='\0';
threshold_out=threshold;
globalerror=globalerrorin;



/* controlla l'architettura */

if (sizeof(short int)!=2 || sizeof(float)!=4 || sizeof(int)!=4)
{
     puts("Sorry, wrong architecture! The program works only on architectures");
     puts("where the following conditions are verified:\n");

     puts("- sizeof(short int)==2");
     puts("- sizeof(float)==4");
     puts("- sizeof(int)==4\n");

     puts("This architecture has:\n");

     printf("- sizeof(short int)==%d\n", sizeof(short int));
     printf("- sizeof(float)==%d\n", sizeof(float));
     printf("- sizeof(int)==%d\n\n", sizeof(int));


     SKIE_out();
     return;
}



/* Se esiste si apre il file header di input */

sprintf(filename, "%s.sdh", basedataset);

input=fopen(filename, "r");
if (!input)
{
     printf("Unable to open file %s.sdh\n", filename);
     fclose(input);
     SKIE_out();
     return;
}


if (fscanf(input, "%d %d %d\n", &data.Dimx, &data.Dimy, &data.Dimz)!=3)
{
     printf("Error reading file %s\n", filename);
     fclose(input);
     SKIE_out();
     return;
}
```

```
    if (fscanf(input, "%f %f %f", &data.Sx, &data.Sy, &data.Sz)!=3)
    {
        printf("Error reading file %s\n", filename);
        fclose(input);
        SKIE_out();
        return;
    }

    if (fscanf(input, "\n%f %f %f", &data.Deltax, &data.Deltay, &data.Deltaz)!=3)
    {
        data.Deltax=0;
        data.Deltay=0;
        data.Deltaz=0;
    }


    fclose(input);


    if(!data.Dimx || !data.Dimy || !data.Dimz || !data.Sx || !data.Sy || !data.Sz)
    {
        printf("Data header (%s) is corrupt or wrong!\n", filename);
        SKIE_out();
        return;
    }

    if (data.Dimx<5 || data.Dimx<5 || data.Dimx<5)
    {
        printf("Data dimensions too small!\n");
        SKIE_out();
        return;
    }


    puts("Emitter v1.0 27-Aug-98\n");
    printf("dataset dimension: %dx%dx%d\n", data.Dimx, data.Dimy, data.Dimz);
    printf("voxel size: %gx%gx%g\n", data.Sx, data.Sy, data.Sz);
    printf("delta values: %gx%gx%g\n", data.Deltax, data.Deltay, data.Deltaz);


    /* calcolo della diagonale come valore informativo */

    diagonal  = (data.Dimx*data.Sx)*data.Dimx*data.Sx;
    diagonal += (data.Dimy*data.Sy)*data.Dimy*data.Sy;
    diagonal += (data.Dimz*data.Sz)*data.Dimz*data.Sz;
    diagonal  = (float)sqrt(diagonal);

    printf("bbox diagonal: %g\n\n", diagonal);

    tot=(data.Dimx<data.Dimy)? data.Dimx : data.Dimz;
    tot=(data.Dimz<tot)? data.Dimz : tot;
    tot/=3;

    if (globalerror<0.0)
    {
        printf("warning: decimation error must be greater equal than 0.0. Using
0.0\n");
        globalerror=0.0;
    }
    else if (globalerror>tot)
    {
        printf("warning: decimation error must be less than %d. Using %d\n",tot,tot);
        globalerror=tot;
    }

    printf("global error: %g\n\n", globalerror);


    if (sizex<2.0 || sizex>data.Dimx)
    {
        printf("warning: 'sizex' value must lie in range [2-%d]. Using default %d.\n",
                data.Dimx, data.Dimx);
        sizex=data.Dimx;
    }
```

```
if (sizey<2.0 || sizey>data.Dimy)
{
     printf("warning: 'sizey' value must lie in range [2-%d]. Using default %d.\n",
               data.Dimy, data.Dimy);
     sizey=data.Dimy;
}

if (sizez<2.0 || sizez>data.Dimz)
{
     printf("warning: 'sizez' value must lie in range [2-%d]. Using default %d.\n",
               data.Dimz, data.Dimz);
     sizez=data.Dimz;
}


/* calcolo delle dimensioni assolute dei blocchi e del numero di slice */


oldsizex=sizex;
oldsizey=sizey;
oldsizez=sizez;


slicex=(int)ceil(((float)data.Dimx-1)/(sizex-1));
slicey=(int)ceil(((float)data.Dimy-1)/(sizey-1));
slicez=(int)ceil(((float)data.Dimz-1)/(sizez-1));


/* se e' presente il file "".cmd allora significa che il dataset e' gia' stato */
/* spacchettato, dunque si possono leggere direttamente i pacchetti se la loro */
/* dimensione e' uguale a quella precedente contenuta nel file "".cmd. */

sprintf(filename, "%s.cmd", basedataset);
input=fopen(filename, "r");

if (input)
{
     int res=fscanf(input, "%d %d %d", &sx, &sy, &sz);
     fclose(input);

     if (res==3 && (sx==oldsizex && sy==oldsizey && sz==oldsizez))
     {

          printf("emitter fast approach: dataset already unpacked in %d files.\n",
                    slicex*slicey*slicez);
          fflush(stdout);


          /* Si spedisce i path relativi ai pacchetti */

          for (sz=0; sz<slicez; sz++)
               for (sy=0; sy<slicey; sy++)
                    for (sx=0; sx<slicex; sx++)
                    {
                         sprintf(extension, "%d-%d-%d", sx, sy, sz);

                         printf("\n\n-> sub-block n. %d (%dx%dx%d):\n",
                                   blockid, sizex, sizey, sizez);
                         blockid++;

                         skie_error=0;
                         SKIE_out();
                         skie_error=1;
                    }

          return;
     }
}



/* si apre il file dei dati */
```

```
        sprintf(filename, "%s.sdsl", basedataset);

        input=fopen(filename, "rb");
        if (!input)
        {
             printf("Unable to load the %s file!\n", filename);
             SKIE_out();
             return;
        }

        printf("Writing %d files...\n", slicex*slicey*slicez);
        fflush(stdout);


        /* allocazione di una fetta della matrice 3D (tagliata lungo l'asse zeta).
           E' grande sizez*DimX*DimY. E' relativa alla grandezza degli short int
           che vale 2. */

        lenslicez=data.Dimx*data.Dimy*sizez;
        volume=(short int *) calloc(lenslicez, 2);

        if (!volume)
        {
             printf("Insufficient memory. Requested memory was %d bytes.\n", lenslicez*2);
             fclose(input);
             SKIE_out();
             return;
        }


        /* Nota: l'ultima iterazione di ciascun ciclo coinvolge un sottovolume piu' piccolo
   */

        for (sz=0; sz<slicez; sz++)
        {

             if(sz!=slicez-1)
             {
                  tot=0;
                  for (n=0; n<lenslicez; n++)
                       tot+=ReadShort(volume+n, input);

                  if (tot!=lenslicez)
                  {
                       printf("Error reading file %s.sdsl.\n", basedataset);
                       fclose(input);
                       free(volume);
                       SKIE_out();
                       return;
                  }
             }
             else
             {
                  /* si legge l'ultima slice (piu' piccola) lungo la zeta */

                  sizez=data.Dimz-sz*(sizez-1);          /* -1 perche' si torna indietro di
   una linea */
                  lenslicez=data.Dimx*data.Dimy*sizez;

                  tot=0;
                  for (n=0; n<lenslicez; n++)
                       tot+=ReadShort(volume+n, input);


                  if (tot!=lenslicez)
                  {
                       printf("Error reading file %s.sdsl.\n", basedataset);
                       fclose(input);
                       free(volume);
                       SKIE_out();
                       return;
                  }
             }
```

```
                out(
                )

$c{

     char meshin[MAX_PATH], meshout[MAX_PATH];


     while (SKIE_in() != SKIE_EOS)
     {
          if (skie_error_out) continue;


          /* si setta l'uscita */

          sprintf(meshin, "%s-%s_r.msh", basedataset_out, extension_out);
          sprintf(meshout, "%s_f.msh", basedataset_out);

          printf("meshjoinraw %s %s\n", meshin, meshout);
          fflush(stdout);


          if (meshjoinraw(meshin, meshout))
               printf("error: problems with meshjoinraw (%s).\n", meshin);


          /* rimuove il file intermedio che contiene la mesh decimata da meshdeci */

          remove(meshin);
     }



     /* POST-PROCESSING */

     /* Si eliminano i vertici ridondanti trasformando la mesh in formato indicizzato */


     sprintf(meshin, "%s_f.msh", basedataset_out);
     sprintf(meshout, "%s.msh", basedataset_out);
     printf("toindmsh %s %s\n", meshin, meshout);
     fflush(stdout);

     if (toindmsh(meshin, meshout))
          printf("Problems with toindmsh (%s).\n", meshin);
     else
          printf("*** Final indexed mesh written ***\n");


     remove(meshin);

}c$
inc(stdio.h, unistd.h)
obj(meshjoinraw.o, toindmsh.o)
end
```

```
/******* F A R M *********/


pragma parallelism degree 4 in iso_reduction_farm;

farm iso_reduction_farm in(
                           char basedataset[MAX_PATH],
                           char extension[MAX_EXT],
                           float threshold_out,
                           float globalerror,
                           int skie_error
                      )
                      out(
```

```
                                    char basedataset_out[MAX_PATH],
                                    char extension_out[MAX_EXT],
                                    int skie_error_out
                               )

          iso_reduction in(basedataset, extension, threshold_out, globalerror, skie_error)
                         out(basedataset_out, extension_out, skie_error_out)

     end farm


     iso_reduction in(
                         char basedataset[MAX_PATH],
                         char extension[MAX_EXT],
                         float threshold_out,
                         float globalerror,
                         int skie_error
                    )

                    out(
                         char basedataset_out[MAX_PATH],
                         char extension_out[MAX_EXT],
                         int skie_error_out
                    )

     $c{
          char command[MAX_PATH];


          strcpy(basedataset_out, basedataset);
          strcpy(extension_out, extension);
          skie_error_out=skie_error;


          if (!skie_error)
          {

               sprintf(command, "surfactor %s-%s.sdsl %s-%s.msh -t %f -m MC",
                    basedataset, extension, basedataset, extension, threshold_out);

               printf("%s\n", command);

               if(system(command))
               {
                    printf("Problems with system(surfactor).\n");
                    skie_error_out=1;
               }
               else
               {

                    sprintf(command, "meshdeci -z -ta -g -r -p%f %s-%s.msh %s-%s_r.msh",
                         globalerror, basedataset, extension, basedataset, extension);

                    printf("%s\n", command);

                    if(system(command))
                    {
                         printf("Problems with system(meshdeci).\n");
                         skie_error_out=1;
                    }


                    /* rimuove il file intermedio che contiene la mesh prodotta da Surfactor
     */

                    sprintf(command, "%s-%s.msh", basedataset, extension);
                    remove(command);
               }

               fflush(stdout);
          }

     }c$
```

```
inc(stdio.h)
end



/*********** M A I N **********

L'applicazione legge un dataset volumetrico memorizzato nel formato sdsl, ricavato
dall'applicazione di una TAC o generato da un processo di simulazione.
Il dataset viene suddiviso in blocchi eguali (tranne al piu' i blocchi al bordo) piu'
piccoli in modo da sfruttare le potenzialita' di una macchina parallela.
Ogni blocco viene elaborato per estrarre una isosuperficie (con il metodo Marching
Cubes) che successivamente viene semplificata tenendo sotto controllo un parametro
globale di errore.
Le mesh ottenute dai sottoblocchi vengono poi fuse in una sola eliminando i vertici
ridondanti presenti ai bordi.

Il formato di input e' costituito da due file di cui un header di tre righe
ascii (.sdh):

  dimx dimy dimz            dimensione del dataset in voxel
  voxelx voxely voxelz      dimensione del voxel (se omogeneo e' 1.0 1.0 1.0)
  shiftx shifty shifz       spostamento del dataset nello spazio
 ...

il secondo file e' binario e contiene i valori del dataset (.sdsl). Nel file la x cicla
piu' rapidamente segue la y ed infine la z:

  <x1-y1-z1>.......<xn-y1-z1>

             ...

  <x1-ym-z1>.......<xn-ym-z1>

              .
              .
              .

  <x1-y1-zw>.......<xn-y1-zw>

             ...

  <x1-ym-zw>.......<xn-ym-zw>


 Il formato di output (binario) e' una mesh indicizzata della forma:

  INDEXED                   tag identificativo del formato (7 char)

  <int>                     numero di vertici (4 byte)
  <float><float><float>     primo vertice (12 byte in tutto)
         ....
  <float><float><float>     ultimo vertice

  <int>                     numero dei triangoli (4 byte)
  <int><int><int>           primo triangolo (12 byte)
         ....
  <int><int><int>           ultimo triangolo


L'applicazione e' un PIPELINE con al suo interno (iso_reduction_farm) una farm.
Argomenti:

 dataset          dataset 3D volumetrico in formato sdsl (short di 2 byte per i dati)

 sizex, sizey, sizez   dimensione assulote dei sotto-blocchi nei quali viene
                       suddiviso il dataset

 threshold             soglia utilizzata dall'algoritmo Marching Cubes per
                       estrarre l'isosuperficie da dataset volumetrico
```

```
   decim_error          errore globale assoluto.
                        E' utilizzato per semplificare la isosuperficie. Valori
                        di 0.01 o 0.1 producono gia' buone decimazioni.

*/


pipe main in(
              char dataset[MAX_PATH],
              int sizex, int sizey, int sizez,
              float threshold,
              float decim_error
            )

         out()


     /* nodo emettitore (dispatcher) */

     sdsl_unpacking in(
                       dataset,
                       sizex, sizey, sizez,
                       threshold, decim_error
                   )

                   out(stream of {
                        char basedataset[MAX_PATH],
                        char extension[MAX_EXT],
                        float threshold_out,
                        float globalerror,
                        int skie_error
                   })


     /* farm che parallelizza l'estrazione dell'isosurperficie e la semplificazione
della */
     /* mesh prodotta */

     iso_reduction_farm in(
                           basedataset,
                           extension,
                           threshold_out,
                           globalerror,
                           skie_error
                      )

                      out(
                           char basedataset_out[MAX_PATH],
                           char extension_out[MAX_EXT],
                           int skie_error_out
                      )


     /* nodo collettore (collector) */

     mesh_packing  in(stream {
                        basedataset_out,
                        extension_out,
                        skie_error_out
                   })

                   out(
                   )


end pipe
```