**Relazione del Progetto di Architetture e Programmazione dei Sistemi di Elaborazione**

**Gruppo: Arcuri Francesco, Mangione Fabrizio, Morrone Stefano, Perna Stefano (PM)**

# Premessa

Il presente elaborato si pone lo scopo di descrivere il progetto didattico assegnato durante il corso di “Architetture e Progettazione dei Sistemi di Elaborazione”, andando a commentare e descrivere le varie fasi di sviluppo e sull’evoluzione del codice.

Si dà come assunto la conoscenza della traccia assegnata, inoltre si fa notare che l’elaborato di relazione è stato prodotto parallelamente allo sviluppo del codice, riporta quindi le fasi del progetto attraverso un una visone locale e non su una visone complessiva del risultato finale.

# Codice in C

Il progetto parte con lo sviluppo dell’algoritmo “fish school search” attraverso un codice in C, utilizzando i file già messia disposizione dai docenti si è elaborato la struttura dell’algoritmo sul file “fss32c.c”. Per spezzare la complessità dell’algoritmo viene realizzato il codice che ne implementa la logica generale nella funzione “void fss(params\* input){…}”. La struttura principale all’interno della funzione fss è realizzata utilizzando funzioni che sono da sviluppare e che sono assegnate ai vari membri del gruppo.

Immagine che contiene testo

Descrizione generata automaticamenteOltre alle variabili in input si è vista nascere la necessità di inserire altre variabili per memorizzare dati globali che servono al di fuori della singola iterazione e per memorizzare i vari risultati parziali. Di seguito sono riportate le variabili aggiunte per l’esecuzione dell’algoritmo.

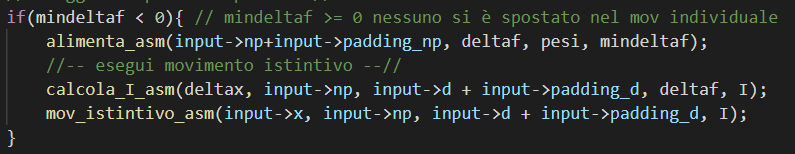
Si passa ora al cuore centrale dell’algoritmo che viene ripetuto fino ad arrivare al numero massimo di iterazioni indicate nell’input. Nelle varie iterazioni si vanno ripetere i vari passi dell’algoritmo, come viene mostrato nel codice C seguente.

Immagine che contiene testo

Descrizione generata automaticamente

La divisione in funzioni ha permesso sia di spezzare la complessità e rendere più leggibile l’algoritmo e di dividere il codice da realizzare tra i componenti del gruppo.

La scrittura della prima versione delle varie funzioni è stata relativamente rapida anche grazie all’aiuto dato dalla traccia dettagliata; infatti, è stato sufficiente seguire le varie operazione aritmetiche dell’algoritmo per realizzare le varie funzioni in C. Una volta combinato il tutto il programma chiaramente non funzionava. Si è quindi resa necessaria una fase di debug in cui si sono identificati gli errori nel codice. Tra gli errori principali identificati quello relativo all’utilizzo eccessivo dei numeri random che andava oltre quelli messi a disposizione dai docenti, dove la difficoltà principale è stata identificare il problema e non la sua risoluzione. Qualche altro piccolo errore è stato identificato nell’accesso alle strutture dati, non per quanto concerne la logica, ma più nell’utilizzo del linguaggio C. Completata questa fase si è arrivati a una prima risoluzione corretta del problema con un tempo netto di … . Questo dato è da considerarsi il punto di partenza da cui seguiranno le successive ottimizzazioni.

In seguito, è stata aggiunta un’ottimizzazione sul codice in C che è la seguente:

se nessun pesce si è spostato tutti i deltaf e tutti i deltax sono pari a zero e quindi le operazioni inserite nel blocco dell’if perdono di significato in quanto non porterebbero a nessuna variazione. Si decide di sostenere il costo di questo confronto per ogni iterazione in quanto le operazioni nell’*if* sono molto onerose è il poterle evitare anche solo poche volte si è visto porta un beneficio ai tempi di esecuzione.

## Ottimizzazioni C

Già nel codice C si sono trovate delle soluzioni allo scopo di ottimizzare l’esecuzione dell’algoritmo. Come prima cosa all’interno delle funzioni che definiscono le fasi principali dell’algoritmo sono state definite nuove funzioni, ad esempio, per il calcolo di numeratore e denominatore (il tutto ha come fine ultimo trovarsi una serie di funzioni facilmente trasformabili in assembly).

Immagine che contiene testo

Descrizione generata automaticamenteUn serie di soluzioni a nostro avviso interessante sono quelle utilizzate nella funzione di “Movimento Individuale” di seguito viene riportato parte del codice.

Si introduce una piccola leggenda per semplificare la lettura del codice

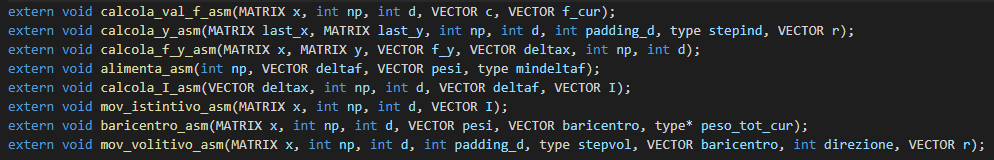
|  |  |
| --- | --- |
| X | Matrice coordinate correnti dei pesci |
| Y | Matrice coordinate dei pesci a seguito del movimento individuale |

Come prevede l’algoritmo stesso i pesci ad essere spostati a seguito del movimento individuale sono solamente quelli che hanno acquisito una posizione migliore (in funzione del tipo di risultato che si sta cercando max/min). L’idea è quello di limitare il numero di scritture sull’array che contiene la posizione dei pesci. Quindi se la maggioranza dei pesci si sono spostati, allora, la matrice Y diventerà la nuova matrice X attraverso uno scambio di puntatori. Ottenere questo risultato è possibile grazie al dato “deltaf” (delta F) di ogni pesce, infatti il pesce i-esimo avrà “deltf” pari a zero se il pesce non si è spostato. Il numero di pesci spostato viene calcolato a ogni movimento individuale del singolo pesce, se il pesce viene spostato questo contatore è incrementato.

Un altro aspetto ad essere sato preso in considerazione e valutato è il seguente: in una fase preliminare si era pensato di aggiornare il valore “f\_min” (valore della funzione minimo) ad ogni spostamento avendo a disposizione il valore di “f” corrente a seguito dello spostamento, andando confrontare il valore minimo corrente (memorizzato in una valribile) con quello appena calcolato, piuttosto che calcolarlo al termine delle iterazioni con un ulteriore ciclo. Ciò risultava molto oneroso a causa della presenza di numerosi salti condizionati ripetuti per ciascuna iterazione. Perciò si è deciso di calcolare il valore ottimo al termine delle iterazioni così da avere i medesimi salti condizionati per una unica funzione che itera sui pesci, invece che per ciascuna delle funzioni presenti e per ciascuna iterazione, così da ottenere una ottimizzazione del codice.

Nelle altre funzioni non si è seguita nessuna ottimizzazione particolare se non quella ci cercare di limitare gli accessi in memoria e limitare le catene di puntatori. Su consiglio del docente si è sempre cercato di utilizzare funzioni *void* e di passare come parametro il valore di ritorno (dove necessario).

# Assembly x86-32+SSE

Terminata la prima fase di stesura dell’algoritmo in C si è passato all’ottimizzazione delle funzioni attraverso la scrittura in codice Assembly. La prima versione che si vuole implementare è per l’architettura “x86-32+SSE”. Per utilizzare il codice assembly è stato necessari apportare alcune modifiche al codice C andando a ridefinire nuove funzioni “extern”, che vanno a sostituire quelle realizzate in C.

Durante lo sviluppo delle varie funzioni in assembly si è pesato di raggiungere un’efficienza maggiore andando ad effettuare il *for* sui pesci direttamente nel codice a basso livello, ma questo non era compatibile con l’utilizzo della direttiva openMP. Si è quindi realizzato per le funzioni:

* calcola\_val\_f\_asm
* calcola\_y\_asm
* calcola\_f\_y\_asm
* mov\_volitivo\_asm

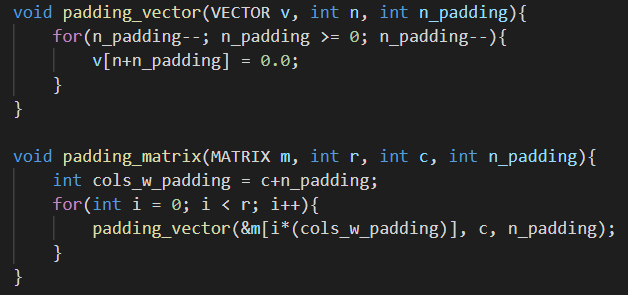
un ulteriore funzione denominata allo stesso modo con l’agginta di omp che appunto è una variante che supporta openMP, ma il *for* sui pesci è realizzato nel codice C e la singola funzione è invocata una volta per ogni pesce.

Immagine che contiene testo

Descrizione generata automaticamentePer le altre funzioni non sopra citate è stata comunque riprodotta una versione identica ma con standard di nomenclatura diverso al fine di trovare coerenza con la procedura compilativa del docente.

Si passa ora quindi alla realizzazione delle varie funzioni con l’obbiettivo di ottimizzare al massimo la velocità di calcolo e limitare gli accessi in memoria, naturalmente con i limiti imposti dal numero di registri.

Come si può notare da queste schermate alcune funzioni prevedono un parametro *padding\_d* che si pone lo scopo di permettere in assembly di utilizzare la operazione di MOVAPS (allineate) così nel caso di rimanenze di elementi non divisibili per gli elementi prelevati in contemporanea sul registro, si può comunque effettuare questa operazione, in quanto per gli elementi mancati sono stati inseriti degli zeri che non influiscono sulle operazioni. In altre funzioni in cui non vi è necessario accedere ai numeri random il padding è già compreso nel parametro *d*, la scelta di divederlo nelle altre funzioni e per evitare di utilizzare numeri random del file del docente per elementi di padding.



Il padding è stato effettuato su ogni vettore il cui numero di elementi non risulta multiplo di 4, e lo si rende tale. Per le matrici sul numero di colonne (numero di coordinate) seguendo la stessa logica, ottenendo così un insieme di elementi che supporta la mov allineata.

Le vari funzioni assembly sono realizzate con dei parametri (costanti) (nel *section .data*) che identifica diversi fattori, alcuni dei quali anche sull’ottimizzazione che efftua, queste costanti sono inserite per rendere il codice più leggibile.

Tecniche di ottimizzazione loop Unroling e loop vetorization

Nell’ottimizzare il codice sono state utilizzate diverse tecniche, tra queste troviamo:

* **Loop Unrolling**: in italiano srotolamento del loop, consiste nella riduzione del numero di salti condizionati andando ad eseguire operazioni ripetute in un ciclo in blocchi di operazioni su un gruppo di elementi del ciclo, così da ridurre le ripetizioni del ciclo stesso. Questa tecnica consente un’ottimizzazione del codice, ma comporta un utilizzo maggiore dei registri.
* **Loop Vectorization**:

## Baricentro

Immagine che contiene testo

Descrizione generata automaticamenteLa funzione baricentro come suggerisce il nome stesso è la funzione che si occupa del calcolo del baricentro.

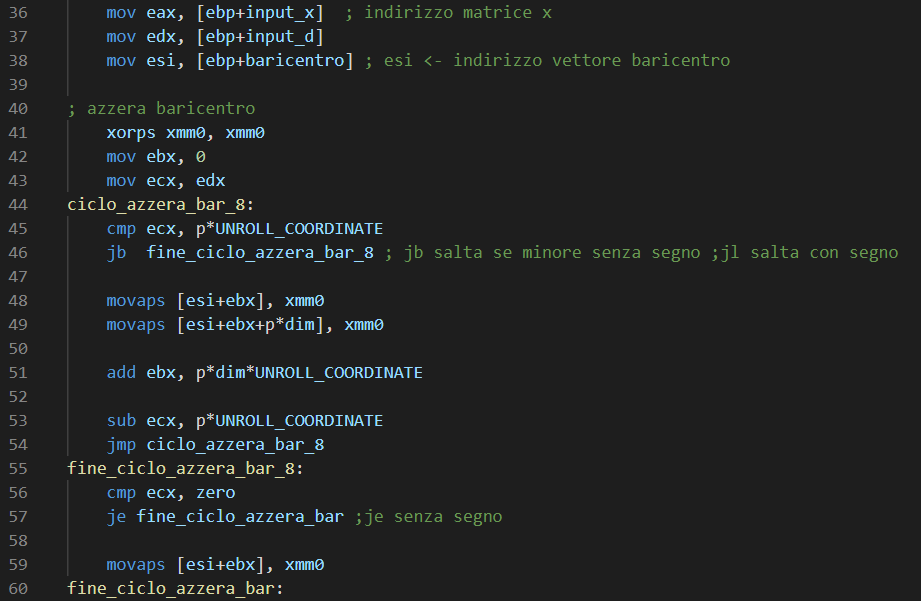
Nella funzione in C si era pensato di dividere il calcolo di numeratore e denominatore in due funzioni separate, ma lavorando in assembly si è notato che si analizzavano dati comuni e questa divisione sarebbe risultata semplificativa dal punto di vista dell’implementazione, ma meno efficiente dal punto di vista computazionale.

Immagine che contiene testo

Descrizione generata automaticamentePer la lettura dei parametri dalla la chiamata da codice C ad assembly si definiscono delle costanti di offset nel “section .tex”. Questo offset rappresenta la posizione progressiva rispetto l’ebp della chiamata di funzione nello stack.

Questa tecnica è stata riproposta per tutte le funzioni in assembly, così a garantire maggiore leggibilità del codice. (la trattazione di questo aspetto non è ripetuta per le funzioni successive).

|  |  |
| --- | --- |
| **eax** | Indirizzo della matrice X |
| **ebx** | contatore pesci (indica il corrente) |
| **ecx** | contatore delle coordinate del pesce corrente |
| **edx** | d – numero di coordinate (\* dim – dimensione degli operandi) |
| **esi** | Indirizzo al vettore baricentro |
| **edi** | np – numero di pesci |



Come si può notare il codice diversi commenti utili sia per l lettura dello stesso che per la correzione in caso di errori. Di seguito saranno riportate le osservazioni principali sul codice sovrastante (si seguirà lo stesso schema per le porzioni a seguire).

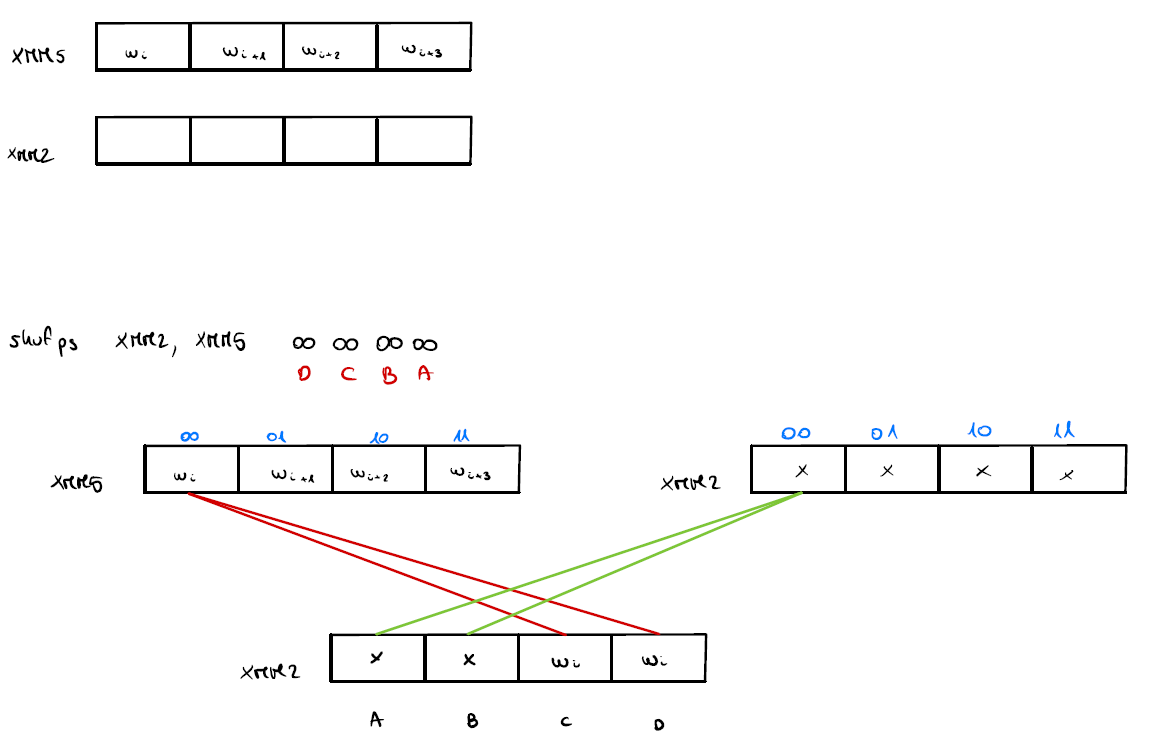
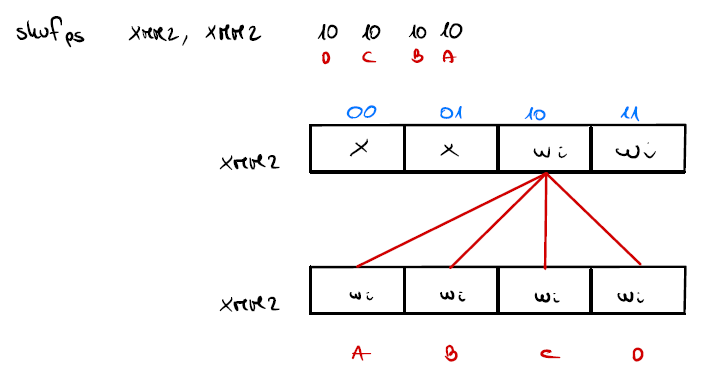
Alla riga 44 inizia un ciclo per compiere delle operazioni di lettura dei dati della memoria e di azzeramento, poiché per calcolare il numeratore abbiamo bisogno del supporto della memoria. Nello specifico quella allocata per il baricentro e al termine avendo il peso totale (calcolato), al termine non ci resta che accedere in memoria e dividere il numeratore per il peso totale.

Nella riga 45 troviamo la costante “p” che sta per packed, ovvero ci indica quanti elementi per volta vengono letti dalla memoria, in questo caso 4 float. “UNROLL\_CORDINATE” indica le volte che viene effettuata l’operazione di unroll.

Immagine che contiene testo

Descrizione generata automaticamente

Alla riga 62 l’operazione di xor di un registro su sé stesso rappresenta l’azzeramento dello stesso. In questo caso si azzera il registro per effettuare la somma totale dei pesi, che viene calcolata mentre si itera sui quattro pesci.

Nella riga 82 come suggerisce il commento stesso si aggiorna la somma parziale dei pesi. Nelle due righe successive l’obbiettivo è quello di andare a selezione il primo dei quattro elementi contenuti in xmm5 e andarlo a replicare sul registro xmm2. Questa operazione è necessaria perché il blocco d’istruzioni che stiamo analizzando è ripetuto quattro volte. L’obbiettivo in fatti di leggere contemporaneamente cinque elementi dal vettore dei pesi dei pesci è quello di ridurre gli accessi in memoria. Di seguito sono riportati dei disegni che chiariscono le righe di codice 83 e 84.

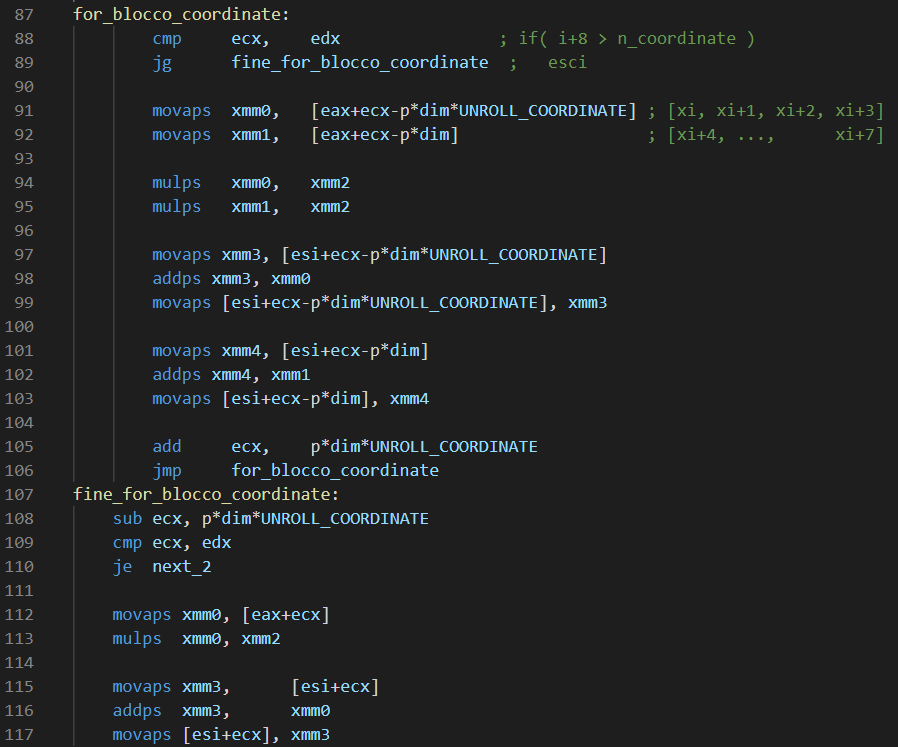
Nella riga 86 si utilizza ecx per il confronto successivo che troviamo a riga 88, il tutto serve per valutare che il numero di coordinate ancora da analizzare sia effettivamente multiplo di 8, e quindi si possa proseguire con questo codice.

Immagine che contiene testo

Descrizione generata automaticamenteNelle righe 91 e 92 due si leggono dalla memoria otto coordinate del pesce i-esimo, mentre nelle righe 94 e 95 si moltiplicano per il peso del pesce. Nei due blocchi di istruzione successivi che vanno da riga 97 a riga 103 si aggiorna il numeratore andando a sommare i valori appena calcolati.

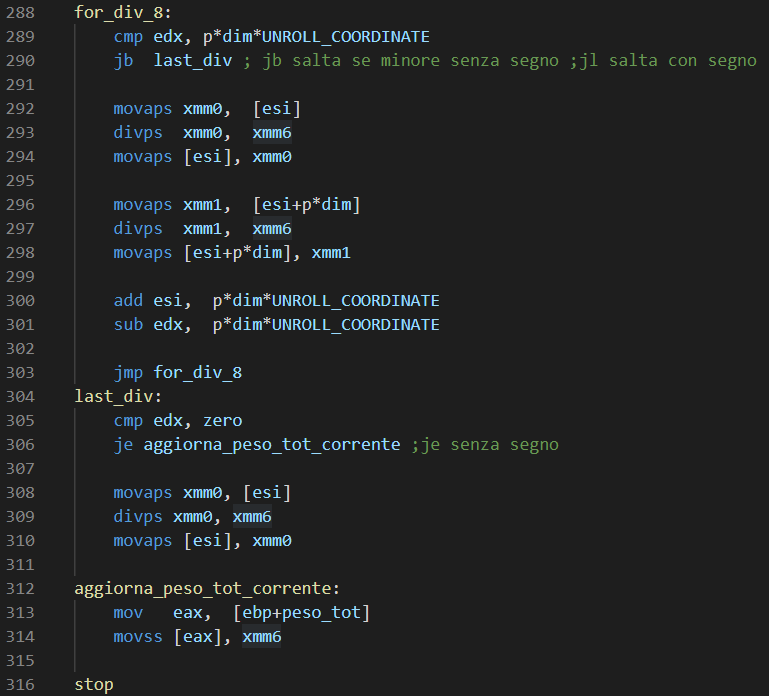
Nella riga 120 come suggerisce il commento stesso si passa all’analisi del secondo pesce e di conseguenza come si può verde nelle due righe successive si utilizza il secondo peso presente in xmm5 e lo si replica su xmm2.

Come anticipato ora inizia una ripetizione del codice precedente sul secondo pesce del blocco da quattro che è viene analizzato, questa operazione non è altro che l’unroll sui pesci.

Come conseguenza di questa gestione a quattro pesci la volta resta da gestire il caso in cui i pesci non siano multipli di quattro e quindi i rimanenti sono analizzati singolarmente. Una volta completata questa serie di operazioni troveremo il numeratore memorizzato nella locazione di memoria del baricentro e ci si può Immagine che contiene testo

Descrizione generata automaticamentecalcolare la somma dei pesi.

Con questa coppia di operazioni calcoliamo il peso totale che si troverà replicato quattro volte nel registro. A questo punto rimane solamente l’operazione di calcolo del baricentro con la divisione tra numeratore e denominatore.



In questa ultima parte di codice si effettua l’operazione sopra annunciata infatti come per le parti precedenti prima si effettua l’operazione sulle coordinate a gruppi di otto elementi ogni salto del ciclo e poi le rimanenti non multiple di 8. Un altro passaggio importante è il salvataggio in memoria del peso totale che servirà per operazioni successive e potrà essere utilizzato senza necessità di ricalcolarlo.