ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – Scienza e Ingegneria DISI

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in

Sistemi Intelligenti M

Utilizzo di metodi di configurazione automatica per un'applicazione di trans-precision computing su piattaforma PULP

CANDIDATO
Marco Donato Torsello

RELATRICE
Prof.ssa Michela Milano

CORRELATORI
Michele Lombardi
Davide Rossi

Anno Accademico 2016/17

Sessione II

Indice

1 Introduzione

Dai datacenter ai nodi dell'Internet of Things il consumo energetico sta diventando sempre di più un argomento centrale, per questo l'industria sta cercando sempre nuovi modi per abbattere l'impatto energetico dei propri prodotti. Un approccio innovativo per risolvere questo problema è quello espresso dal concetto di Transprecision Computing, che punta ad usare le metodologie dell'Approximate Computing ad un granularità più fine. In questo modo, aumenta l'adattabilità del metodo alle diverse situazioni che si possono presentare nelle applicazioni. L'obiettivo di questa tesi è quello di mostrare come sia possibile sfruttare metodologie di configurazione automatica su un'applicazione di Transprecision computing, il tutto utilizzando la piattaforma PULP.

1.1 Approximate Computing

I sistemi di calcolo tradizionali per molti anni si sono basati sulla precisione garantita del calcolo per ogni singolo step. Questo approccio si basava sulla legge di Moore e sul miglioramento esponenziale dell'efficienza di calcolo. Negli ultimi anni, tuttavia, l'avvento dell'era dei Big Data ha portato l'urgenza di trovare una soluzione alternativa al calcolo tradizionale, che comporti un consumo energetico dell'ordine di picojoule a operazione.

L'Approximate Computing si basa sul concetto che non è sempre necessario utilizzare la precisione massima in ogni singolo step di calcolo, ma a volte è sufficiente utilizzare un' approssimazione per ottenere un risultato che sia considerato accettabile. Rinunciando alla precisione del risultato step by step, il processo di calcolo si snellisce notevolmente, permettendo di contenere il consumo energetico e allo stesso tempo di raggiungere un risultato accurato. Un esempio classico alla portata di tutti di Approximate Computing è quello dell'utilizzo della compressione lossy per le immagini e altri dati multimediali.

1.1.1 Panoramica delle tecniche di Approximate Computing

Esistono diverse tecniche che permettono di abbattere il consumo energetico e migliorare le performance di calcolose si è disposti a sacrificare una risposta "perfetta".

Queste tecniche possono essere classificate per tipo, visibilità, testabilità e granularità in una ben precisa tassonomia. Per quanto riguarda il tipo di tecnica è possibile lavorare a livello di hardware o di

software. Quando ci si riferisce alla visibilità, invece, si dice che una tecnica ha visibilità computazionale quando si riesce a risalire a quale istruzione ha introdotto l'errore, mentre ha visibilità di tipo dati quando non si può. Un esempio di visibilità di tipo dati è il caso della Low-Refresh DRAM: con questa tecnica non si effettua il refresh della memoria con la stessa frequenza richiesta in condizioni normali, il che può causare un possibile bit-switch nei dati. In questo caso, l'errore può avvenire in qualsiasi momento, completamento slegato dalle istruzioni che vengono eseguite in quel momento.

Una tecnica ha una testabilità deterministica quando, dato lo stesso stato iniziale per tutti gli input, si ottiene sempre lo stesso errore, non deterministica nel caso contrario. Un esempio che si può fare è la differenza che hanno le due tecniche: Syncronization Elision e Code Perforation. Con la prima tecnica le primitive di sincronizzazione vengono evitate, esponendo il programma a casi di corse critiche sui dati condivisi. Nel secondo caso, invece, alcune istruzioni all'interno del codice vengono ignorate. Testare un algoritmo che sfrutta la Syncronization Elision su diverse macchine potrebbe portare a risultati diversi, mentre la Code Perforation porta sempre alla stesso risultato.

La granularità di una tecnica indica se è applicabile solo a piccole sezioni di codice oppure totalmente al sistema. Un caso a granularità grossa è quello della tecnica nota come fuzzy function memoization (1), in cui vengono riutilizzate i risultati di operazioni già avvenute, mentre la fuzzy floating-point instructions (2) ha una granularità fine.

Nella seguente tabella sono presenti alcune delle possibili tecniche utilizzate nell'ambito dell'Approximate Computing.

Tecniche software	Visibilità	Testabilità	Granularità
Approximate CUDA Kernel	**Computazionale**	**Deterministica**	**Grossa**
Approximate Synthesis	**Computazionale**	**Deterministica**	**Grossa**
Algorithm Selection	**Computazionale**	**Deterministica**	**Grossa**
Code Perforation	**Computazionale**	**Deterministica**	**Grossa**
Parallel Pattern Replacement	**Computazionale**	**Deterministica**	**Grossa**
Bit-width Reduction	**Computazionale**	**Deterministica**	**Fine**
Float-to-fixed Conversion	**Computazionale**	**Deterministica**	**Fine**
Approximate Parellelization	**Computazionale**	**Non deterministica**	**Grossa**
Statistical Query	**Computazionale**	**Non deterministica**	**Grossa**
Syncronization Elision	**Computazionale**	**Non deterministica**	**Grossa**
Lossy Compression / Packing	**Dati**	**Deterministica**	**Grossa**

Tecniche hardware	Visibilità	Testabilità	Flessibilità
Digital Neural Accelleration	**Computazionale**	**Deterministica**	**Grossa**
Interpolated Memoization	**Computazionale**	**Deterministica**	**Grossa**
Load Value Approximation	**Computazionale**	**Deterministica**	**Fine**
Instruction Memoization	**Computazionale**	**Deterministica**	**Fine**
Precision Scaling	**Computazionale**	**Deterministica**	**Fine**
Logical Simplification	**Computazionale**	**Deterministica**	**Fine**
Reduced-Precision FPU	**Computazionale**	**Deterministica**	**Fine**
Analog Neural Accelleration	**Computazionale**	**Non deterministica**	**Grossa**
Approx. Precessors	**Computazionale**	**Non deterministica**	**Fine**
Voltage Overscaling	**Computazionale**	**Non deterministica**	**Fine**
Approx. PCM Multi-Level Cells	**Computazionale**	**Non deterministica**	**Fine**
SRAM Soft-Error Exposure	**Computazionale**	**Non deterministica**	**Fine**
Approximate Value Deduplication	**Dati**	**Deterministica**	**Grossa**
Approx. PCM Failed Cells	**Dati**	**Non deterministica**	**Fine**
Low-Refresh DRAM	**Dati**	**Non deterministica**	**Fine**

Tabella 1 Tassonomia delle tecniche di Approximate Computing (3)

La tecnica che verrà utilizzata in questa tesi consiste in una variante della Bit-Width Reduction: questa tecnica consiste nel ridurre il numero di bit usati per rappresentare un numero in virgola mobile, rispetto a quello utilizzato nello standard. La differenza rispetto al caso tradizionale è che, invece di permettere qualsiasi valore di bit per la mantissa, sono ammessi solo lunghezze di mantissa che corrispondono allo standard per i tipi float32, float16 e float8.

La tecnica utilizzata è, quindi, di tipo software, computazionale, deterministica e non propriamente fine.

Nell'applicazione sono presenti sette funzioni, su ognuna di queste verrà applicata la nostra variante di Bit-Width Reduction, scegliendo la quantità di bit di mantissa da utilizzare per ogni variabile..

Rappresentazione dei numeri floating point

Per rappresentare numeri reali, tradizionalmente, sono possibili due modalità: a virgola mobile e a virgola fissa. La modalità a virgola mobile è quella tradizionalmente più utilizzata, è analoga alla notazione scientifica e come tale separa il numero in due parti, la mantissa e l'esponente. In aggiunta a queste due componenti è rappresentato separatamente anche il segno. Lo standard riconosciuto per la rappresentazione dei numeri in virgola mobile, è definito da IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) (4). Secondo lo standard, un numero è formato di 32, 64 o 128 bit a seconda della precisione voluta. In tutti i casi il numero per essere rappresentato è diviso in tre parti: segno, esponente e mantissa. Nel

caso del float in C (32 bit) viene utilizzato un bit per rappresentare il segno, otto per rappresentare l'esponente ed i rimanenti per contenere la parte decimale. Lo standard della IEEE prevede, inoltre, i metodi con cui si dovrebbero applicare gli arrotondamenti.

La rappresentazione fixed, invece, è molto più semplice: il numero è un intero a cui viene applicato un fattore di scala (Ad esempio il numero 4,396 può venire rappresentato come 4396 applicando un valore di scala pari a 1000).

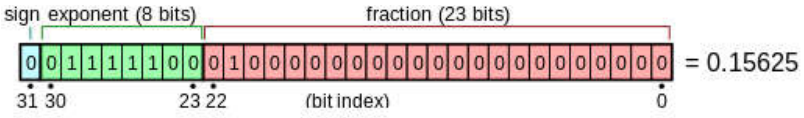


Figura 1 Rappresentazione di un numero floating point a 32 bit (5)

Esistono quattro possibili dimensioni che si possono esplorare per applicare un criterio di approssimazione:

- Riduzione dei bit della mantissa
- Riduzione dei bit dell'esponente
- Cambio della radice implicita
- Semplificazione dei metodi di arrotondamento

Per valutare la solidità dell'approccio di Approximate Computing sono stati effettuati degli esperimenti preliminari, in cui è stata verificata la capacità di ottenere un risultato accettabile apportando una serie di semplificazioni al metodo tradizionale di calcolo. Dai risultati di un esperimento (6) su quattro applicazioni (Sphinx III, ALVINN, PCASYS,

Bench22), in cui si riducono gradualmente il numero di bit di mantissa ed esponente partendo dai valore definiti dallo standard, si è visto che:

- Il numero di bit della mantissa non ha un effetto tangibile sull'accuratezza dei software fino a che non si arriva ad utilizzare solo nove bit.
- Il numero di bit dell'esponente non inficia l'accuratezza fino a che non cala sotto i sette bit, a tal punto l'accuratezza dell'applicazione crolla in tutti i casi, in alcuni casi rendendo impossibile concludere l'esecuzione correttamente.

Successivamente, nello stesso esperimento, è stato analizzato il comportamento di un unità a virgola mobile che possa lavorare con mantisse ed esponenti di lunghezza variabili, e si è trovato che il consumo energetico è linearmente proporzionale al numero di bit degli operandi, cosa che è vera anche per la latenza delle operazioni. Unicamente nel caso a 24 bit, però, la floating point unit consuma il 40% in più di una tradizionale.

1.1.2 Minimizzazione del consumo energetico causato dalle operazioni floating point

I numeri floating point permettono di rappresentare un vasto range di valori reali senza che lo sviluppatore si debba preoccupare di scalarli manualmente, cosa che invece accade quando si utilizza una rappresentazione a virgola fissa. Al contempo, tuttavia, l'hardware che implementa queste tipo di operazioni è particolarmente energivoro.

L'utilizzo di dati fixed point renderebbe lo sviluppo di alcune applicazioni, come quelle di voice recognition, pattern matching e signal processing, molto più complicato; è quindi prassi utilizzare variabili floating point, che, inoltre, permettono un range di dati rappresentabili molto maggiore rispetto a quelli a virgola fissa (6).

Alcune applicazioni floating point riescono ad ottenere un risultato più che accettabile anche quando utilizzano sensori a bassa risoluzione. Poiché è possibile utilizzare il minor numero possibile di bit per la rappresentazione dei numeri, si potrebbe pensare di modificare l'hardware necessario per l'utilizzo di numeri in virgola mobile, ottenendo così un notevole risparmio energetico, senza ledere l'accuratezza dell'applicazione.

1.2 Il progetto OPRECOMP

Open TransPREcision COMPuting (7) è un progetto sviluppato in collaborazione tra realtà industriali e Università, che mira ad implementare il concetto di Transprecision Computing in diverse applicazioni. L'idea alla base è che mettendo in pratica il concetto di approssimazione sia dal punto di vista hardware che software si possa raggiungere un enorme risparmio nel consumo energetico senza che la qualità del risultato peggiori.

Il Transprecision Computing ha come obiettivo quello di abbandonare l'astrazione della "precisione numerica garantita" e sostituirla con una più flessibile ed efficiente. Come idea è chiaramente legato al concetto di Approximate Computing, ma il primo punta a espandere ad approfondire lo stato dell'arte di quest'ultimo. Infatti, il Transprecision

Computing, oltre alla già nota ricerca dell'approssimazione numerica, include uno studio della teoria matematica e dei principi fisici alla base dell'approccio computazionale, così come la progettazione di algoritmi, circuiti e strumenti software ad hoc per la realizzazione di un complesso sistema capace di essere utilizzato in diverse applicazioni. L'obiettivo, infatti, è di realizzare un sistema versatile, che abbia il potenziale di essere sfruttato in una larga varietà di problemi reali, anche non limitati dalla potenza di calcolo, che possano trarre vantaggio dalla approssimazione e accuratezza del metodo stesso. Per la prima volta, il Transprecision Computing, raccoglie know-how relativi a diverse branche del sapere, dalla programmazione alle scienze dei materiali, al fine di ottenere un sistema non solo approssimativo, cioè capace di sveltire le operazioni, ma anche flessibile, robusto, capace di imparare, che vada a mimare i funzionamento di una rete neurale.

OPRECOMP mira a rendere possibile un controllo che permetta di indicare dove e come verranno utilizzate diverse strategie di approssimazione, a livello sia hardware che software, per controllare il comportamento delle approssimazioni. Inoltre, vuole permettere di controllare la precisione di operazioni intermedie non solo in applicazioni in cui il requisito end-to-end non sia stringente, ma anche dove questi requisiti siano vincolanti. Infine, vuole prendere inspirazione dalla natura per creare architetture che permettano di superare i colli di bottiglia causati da quella Von Neumann.

Per fare questo si vuole dimostrare la potenzialità di questo approccio in due scenari reali, uno nell'ordine dei milliwatt e l'altro in quello dei

kilowatt, in entrambi i casi si vuole raggiungere un miglioramento nell'efficienza energetica almeno nell'ordine della magnitudine.

1.3 PULP

Una delle piattaforme che OPRECOMP utilizza per analizzare il suo apporto è quella di PULP, sviluppata in collaborazione tra il Politecnico federale di Zurigo e l'Università di Bologna. PULP, che sta per Parallel Ultra-Low- Power Processor, è un progetto open-source che mira ad ottenere la più alta efficienza possibile in più scenari possibili. PULP è una piattaforma multi-core a bassissimo consumo energetico progettata per soddisfare i requisiti delle applicazioni IoT. A differenza dei microcontrollori a singolo core, il suo parallelismo permette di soddisfare i requisiti di queste applicazioni mantenendo il consumo energetico nell'ordine dei milliwatt. Questa piattaforma è il target dell'applicazione che è stata presa in esame.

2 L'applicazione

Tra le diverse applicazioni che hanno come target la piattaforma PULP, per questo lavoro di tesi è stata selezionata Seizure Detection, un esempio di utilizzo in campo clinico del Transprecision Computing.

L'applicazione che è stata scelta per questa analisi permette di rilevare la presenza di un attacco epilettico analizzando in tempo reale le elettroencefalografie ricavate da una serie di sensori applicati al paziente.

I dati utilizzati dall'applicazione sono stati ottenuti durante una ricerca svolta al Children's Hospital di Boston (8).

L'applicazione effettua ad ogni finestra di dati di input, pari ad un secondo di campionamenti, una Principal Component Analysis, su cui viene eseguita una trasformata discreta Wavelet. Il risultato viene poi inserito in una Support Vector Machine, il cui modello, precedentemente addestrato, è in grado di riconoscere l'assenza o la presenza di un attacco.

2.1 Strumenti utilizzati

2.1.1 Principal Component Analysis

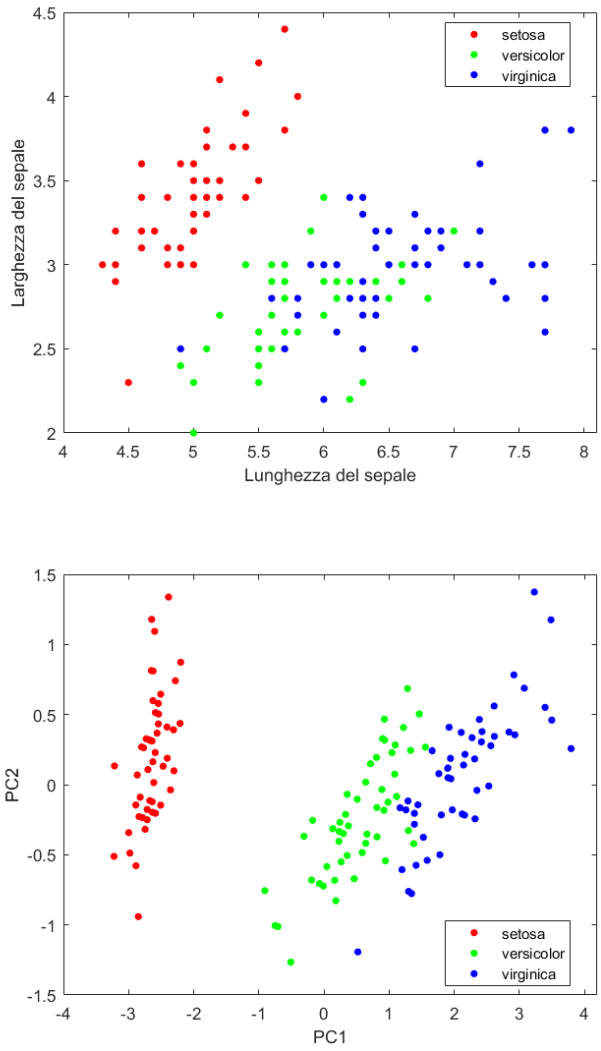


Figura 2 Esempio d'utilizzo di PCA su Dataset Iris (Sopra i dati originali e sotto dopo l'analisi)

Principal Component Analysis è un processo di riduzione di dimensionalità, è in grado quindi di ridurre il numero di variabili da prendere in esame per svolgere le analisi successive.

In maniera intuitiva si può immaginare di creare un ellissoide in grado di contenere tutti i dati, dove ogni asse dell'ellissoide rappresenta un componente principale. Se l'asse dell'ellissoide è corto vuol dire che anche la varianza rispetto quell'asse sarà piccola e che, omettendo l'asse, la quantità di dati omessi è trascurabile (9). In questo modo è possibile selezionare solo le prime variabili risultanti in termini di PCA, ovvero quelle con varianza maggiore. In questo modo è possibile ridurre il numero di variabili rispetto a quelle originali e conservare le più significative, senza che la qualità dei risultati successivi peggiori drasticamente.

Per trovare gli assi dell'ellissoide è necessario centrare i dati nell'origine, a questo punto viene calcolata la matrice delle covarianze e i suoi autovalori e autovettori. Ognuno degli autovettori ortogonali è normalizzato per diventare un vettore unità. Successivamente, ognuno degli autovettori mutualmente ortogonali può essere considerato come un asse dell'ellissoide. La varianza rappresentata dal vettore può essere calcolata come l'autovalore corrispondente dell' autovettore, diviso per la sommatoria di tutti gli autovalori.

2.1.2 Discrete Wavelet Transform

Le wavelet vengono utilizzate per la rappresentazioni di segnali mediante l'uso di una forma d'onda oscillante di lunghezza finita, nota come wavelet madre (10).

Una trasformata wavelet discreta è una trasformata wavelet per cui le wavelet sono discretamente campionate (11).L'utilizzo di una Discrete Wawelet Transform permette di rappresentare un segnale in una forma

più ridondante. Un vantaggio fondamentale, inoltre, rispetto una trasformata di Fourier (una possibile alternativa a una trasformata Wavelet), dove i segnali sono rappresentati come somme d'armoniche, è che le wavelet sono localizzate sia nel tempo che nella frequenza, cosa che non avviene nel caso di Fourier (10). Infine, la prima ha una complessità computazionale pari O(N) mentre la seconda, più lenta, O(N log N).

2.1.3 Support Vector Machine

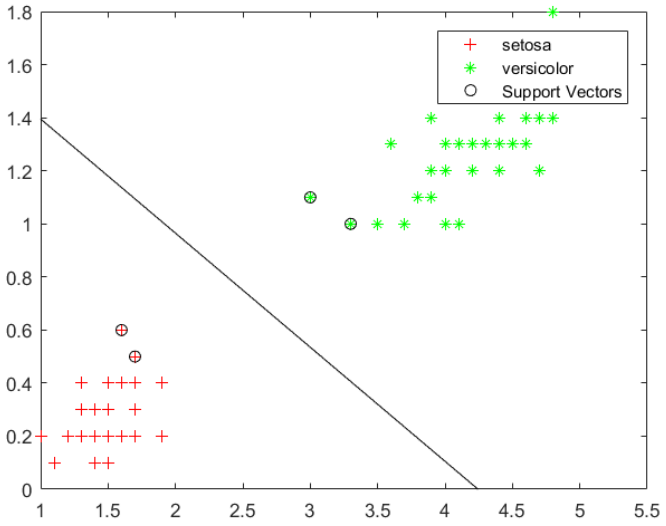


Figura 3 Esempio d'utilizzo di SVM su una versione modificata del dataset Iris

Le Support Vector Machines sono uno strumento di machine learning utilizzato per la classificazione e la regressione (12). Dato un training set in cui i dati sono marcati come appartenenti ad una classe o ad un'altra, questo strumento è in grado di costruire un modello che possa riconoscere a quale classe appartenga un nuovo oggetto in ingresso. Per

fare questo, il modello è costituito da un iperpiano che separa la spazio contente i dati in due semispazi, in cui sono isolati i dati delle due classi.

2.2 Il modello MATLAB

Per generare il modello delle Support Vector Machines è stato utilizzato questo programma a partire dai dati ottenuto dalla precedente ricerca (8).

```
load('datiInput.mat');     % Si carica la matrice dei dati acquisiti e il vettore delle label
load('label.mat');       % Questo vettore serve specificare se è presente un attacco o meno alla rispettiva finestra

x=x';                    % Si effettua la trasposta della matrice x

[Y,X1,d,Psi]=pcanew(x);  % Si chiama la funzione pcanew che effettua la riduzione di dimensionalità
                         %    Si    ottiene    (in    ordine)    gli autovettori, i
                         % componenti principali, gli autovettori e i valori
                         % medi

[PC c] = size(X1);
window_lenght=256;       % La frequenza di campionamento è pari a 256 Hz

% Si effettua una DWT di tipo Haar, per ricavare la matrice delle energie
window=c/window_lenght;
wname = 'haar';
x_new=[];

for i=1:PC
    for j=0:window-1
        if j==0
            dwt = mdwtdec('r',X1(i,1:window_lenght),4,wname);
            [r1,c1]=size(dwt.cd{1,1});
            temp1=power(dwt.cd{1,1},2);
            [r2,c2]=size(dwt.cd{1,2});
            temp2=power(dwt.cd{1,2},2);
            [r3,c3]=size(dwt.cd{1,3});
            temp3=power(dwt.cd{1,3},2);
            [r4,c4]=size(dwt.cd{1,4});
            temp4=power(dwt.cd{1,4},2);
            CD(1,1:window_lenght)=(sum(temp1));
            CD(2,1:window_lenght)=(sum(temp2));
```

```
            CD(3,1:window_lenght)=(sum(temp3));
            CD(4,1:window_lenght)=(sum(temp4));

            x_new1(1:window_lenght,:)=CD';

        else

                dwt                                                 =
mdwtdec('r',X1(i,(j)*window_lenght:((j+1)*window_lenght)-
1),4,wname);
                [r1,c1]=size(dwt.cd{1,1});
                temp1=power(dwt.cd{1,1},2);
                [r2,c2]=size(dwt.cd{1,2});
                temp2=power(dwt.cd{1,2},2);
                [r3,c3]=size(dwt.cd{1,3});
                temp3=power(dwt.cd{1,3},2);
                [r4,c4]=size(dwt.cd{1,4});
                temp4=power(dwt.cd{1,4},2);
                CD(1,1:window_lenght)=(sum(temp1));
                CD(2,1:window_lenght)=(sum(temp2));
                CD(3,1:window_lenght)=(sum(temp3));
                CD(4,1:window_lenght)=(sum(temp4));


x_new1(((j)*window_lenght):((j+1)*window_lenght)-1,:)=CD';

        end
    end
    x_new=[x_new,x_new1];
end
% Si effettua la classificazione usando SVM
EXP_COEFF=0.99;
SCALA=10^(-5);
% In questo modo si utilizza il primo attacco per il training e
il resto per il testing
TEST_PERC=0.28;

dati_train=[];
label_train=[];
dati_test=[];
label_test=[];

[PC c] = size(x_new');
dati_train=x_new(1:10:round(c*TEST_PERC)+10000,:);
label_train=label(1:10:round(c*TEST_PERC)+10000,1);
dati_test=x_new(round(c*TEST_PERC)+1+10000:10:c,:);
label_test=label(round(c*TEST_PERC)+1+10000:10:c,1);
% I dati vengono scalati
dati_train=(dati_train*SCALA);
dati_test=(dati_test*SCALA);
% Viene utilizato un filtro esponenziale
dati_train=exp_filter(EXP_COEFF,dati_train);
dati_test=exp_filter(EXP_COEFF,dati_test);
% Viene creato il modello utilizzando la libreria libSVM (13)
model = svmtrain( label_train,dati_train,'-c 1 -h 0');
```

```
% Viene svolta la  classificazione dei dati di test
[predicted_label,  acc,  decision_val]  =  svmpredict(label_test,
dati_test, model);
```

Dall'encefalogramma di un paziente vengono selezionate tre differenti acquisizione di dati (di solito un'acquisizione dura un'ora, ed ha una frequenza di campionamento pari a 256 Hz) nelle quali avviene almeno un attacco. Un'acquisizione viene utilizzata per la fase di learning e le altre due per il testing. Si utilizza PCA per effettuare una riduzione di dimensionalità, il risultato viene poi processato dalla DWT. I dati risultanti dalla trasformazione wavelet sono poi scalati e separati in due gruppi, il primo è utilizzato per il training e il secondo per il testing col modello risultante.

Per l'implementazione di SVM è stata usata la libreria libSVM (13).

2.3 L'applicazione su piattaforma PULP

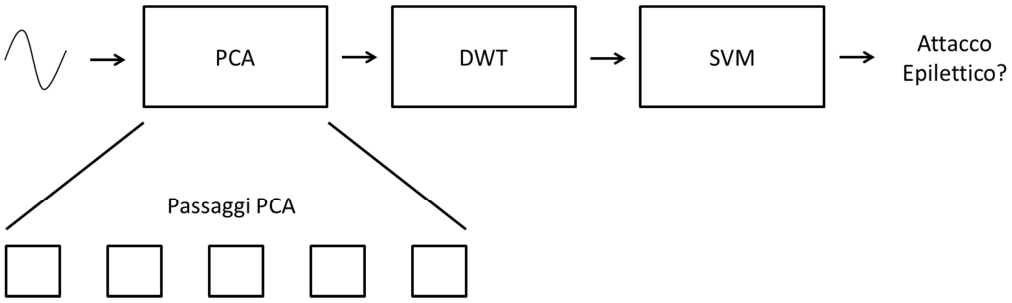


Figura 4 Architettura generale dell'applicazione

Il modello ottenuto da MATLAB è stato poi utilizzato per creare l'applicazione su piattaforma PULP.

Il comportamento di questa versione è identico a quello MATLAB: i dati in ingresso passano attraverso una PCA, per poi passare in DWT ed

infine vengono classificati grazie una SVM, che condivide lo stesso modello con la versione originaria.

L'implementazione del componente DWT deriva da quella presente nella GNU Scientific Library (14), invece il componente SVM utilizza la libreria libSVM (13), mentre il componente PCA è interamente originale.

Il funzionamento della PCA può essere separato in cinque funzioni (in ordine): mean_covariance, householder, accumulate, diagonalize e PC. Nei seguenti capitoli sarà spiegato come si è voluto investigare il comportamento di questi sette kernel (cinque della PCA e quelli che riguardano DWT e SVM), al variare del numero di bit usati per la mantissa.

3 SMAC

Prima di andare a illustrare lo svolgimento degli esperimenti è necessario spiegare cosa è e come funziona lo strumento che sarà il protagonista degli stessi.

Nei grandi algoritmi sono presenti un numero sempre maggiore di parametri che possono essere utilizzati per configurali e per migliorarne le prestazioni. Un esplorazione manuale di questo tipo è tediosa e potrebbe portare a risultati insoddisfacenti, per questo motivo sono nati diversi strumenti che svolgono questo lavoro in automatico. Esistono metodi che si basano sulla creazione di un modello di regressione che verrà utilizzato per predire le performance del software e per la sua ottimizzazione, uno di questi è SMBO. Sequential Model-Based Optimization (SMBO) (15) si sviluppa in due fasi, sviluppare un modello e ottenere nuovi dati basati sul modello stesso. Il training set di SMBO è così composto $\{(\theta_1,o_1),\ldots,(\theta_n,o_n)\}$, dove la configurazione dei parametri è θ_i e o_i è la performance dell'algoritmo ottenuta con la configurazione θ_i.

Input : Target algorithm A with parameter configuration space $\boldsymbol{\Theta}$; instance set Π; cost metric $\hat{c}$
Output : Optimized (incumbent) parameter configuration, $\boldsymbol{\theta}_{inc}$

$[\mathbf{R}, \boldsymbol{\theta}_{inc}] \leftarrow$ *Initialize*$(\boldsymbol{\Theta}, \Pi)$
repeat
  $[\mathcal{M}, t_{fit}] \leftarrow$ *FitModel*$(\mathbf{R})$
  $[\vec{\boldsymbol{\Theta}}_{new}, t_{select}] \leftarrow$ *SelectConfigurations*$(\mathcal{M}, \boldsymbol{\theta}_{inc}, \boldsymbol{\Theta})$
  $[\mathbf{R}, \boldsymbol{\theta}_{inc}] \leftarrow$ *Intensify*$(\vec{\boldsymbol{\Theta}}_{new}, \boldsymbol{\theta}_{inc}, \mathcal{M}, \mathbf{R}, t_{fit} + t_{select}, \Pi, \hat{c})$
until *total time budget for configuration exhausted*
return $\boldsymbol{\theta}_{inc}$

Figura 5 Algoritmo SMBO

In Figura 5 è indicato il funzionamento dell'algoritmo di SMBO: i parametri d'ingresso sono tre, l' algoritmo da ottimizzare con lo spazio di configurazioni di parametri, l'insieme delle istanze su cui verranno provate le configurazioni e la metrica di costo che si tenta di ottimizzare. Il risultato restituito è la migliore configurazione dei parametri ottenuta. Il ciclo principale è composto da tre parti, all'inizio, a partire dal dataset, si crea un modello che poi verrà utilizzato per trovare delle nuove configurazioni promettenti, seguirà una fase d'intensificazione che rimpolperà il dataset con nuovi dati. Questo procedimento si ripete finché il budget di tempo non si esaurisce.

SMBO è un framework generale che viene derivato in due varianti: ROAR e SMAC. Random Online Aggressive Racing (ROAR) è una specializzazione di SMBO che non utilizza un modello ,ma che ogni volta sceglie a caso una nuova configurazione da confrontare con quella attualmente considerata migliore. ROAR è risultato sorprendentemente efficace, ma non è altrettanto interessante come SMAC.

Il modello usato da Sequential Model-based Algorithm Configuration(SMAC) è basato su random forest, uno strumento usato in machine learning sia per classificazione che per la regressione. Una random forest è un insieme di regression tree, che sono simili a gli alberi decisionali, ma sulle cui foglie, al posto degli identificatori delle classi, si trova la performance prevista dell'algoritmo.

Per selezionare una configurazione promettente, il sistema calcola l'Expected Positive Improvement, EI(θ), rispetto alla migliore configurazione vista fino a quel momento, incumbent. EI(θ) è alta per le

configurazioni che hanno un valore predetto basso e per quelle per cui la predizione è incerta. Questo permette che l'algoritmo passi in rassegna sia le parti dello spazio riconosciute come buone (exploitation) sia l'esplorazione di parti ancora non conosciute (exploration).

Per trovare delle configurazioni promettenti, utilizzando il modello, si usa una strategia multi-start local search: vengono selezionate le configurazioni con EI(θ) più alto e si svolge una ricerca locale per ognuna di queste. Per fare questo vengono prese anche le configurazioni che differiscono per un solo parametro dalla configurazione di partenza, nel caso di un parametro categorico viene presa solo una nuova configurazione, mentre nel caso numerico si prendono quattro nuove configurazioni. La ricerca si ferma quando non si trova una configurazione che comporta il miglioramento dell' EI(θ). Oltre alle configurazioni così trovate si aggiungono un numero arbitrario di configurazioni campionate casualmente nello spazio delle configurazioni.

Dopo aver individuato quali configurazioni sono considerate interessanti, si procede con l'intensificazione. L'intensificazione ha il compito di indicare quante valutazioni svolgere per ogni configurazione, decidere quando una configurazione può venire considerata affidabile per essere definita il nuovo incumbent e quali istanze usare per ogni esecuzione. Questa procedura prende in considerazione le nuove configurazioni promettenti, provenienti dal precedente passaggio, e le confronta con la migliore configurazione attualmente presa in esame fino a che il tempo messo a disposizione dell'algoritmo non si esaurisce.

Per ogni confronto tra la nuova configurazione, θ_{new}, e l'incumbent, θ_{inc}, viene eseguito una nuova esecuzione di quest'ultimo per un istanza scelta casualmente, successivamente vengono eseguite nuove esecuzioni per il nuovo contendente, finche i risultati ottenuti sono più bassi rispetto a quelli del secondo, oppure il numero di esecuzioni dei due è uguali e i risultati del primo siano non minori rispetto a quelli del secondo, ed in questo caso θ_{new} diventa il nuovo θ_{inc}.

Questo algoritmo si ripete fino a che il tempo a disposizione non viene terminato.

3.1 Regression Tree

I Regression Tree sono un noto strumento di machine learning, in questa sezione si cercherà di spiegare il loro funzionamento.

Parametro 1	Parametro 2	Parametro 3	Risultato
Falso	**2**	**Rosso**	**3.7**
Falso	**2.5**	**Blu**	**20**
Vero	**5.5**	**Rosso**	**2.1**
Falso	**5.5**	**Blue**	**25**
Falso	**5**	**Rosso**	**1.2**
Vero	**4.5**	**Verde**	**19**
Vero	**4**	**Blu**	**12**
Vero	**3.5**	**Verde**	**17**

Tabella 2 Dati d'esempio Regression Tree (16)

A partire dai dati della Tabella 2 si può vedere come si può generare un Regression Tree che sia in grado di predire il risultato a partire dai parametri di una nuova istanza del problema. In ogni nodo interno

dell'albero viene memorizzato il criterio di discernimento utilizzato, mentre nei nodi foglia è memorizzato il risultato previsto. Per prevedere il risultato di una nuova istanza si percorre l'albero dalla radice fino ad una foglia, seguendo ad ogni nodo interno il ramo la cui condizione viene soddisfatta.

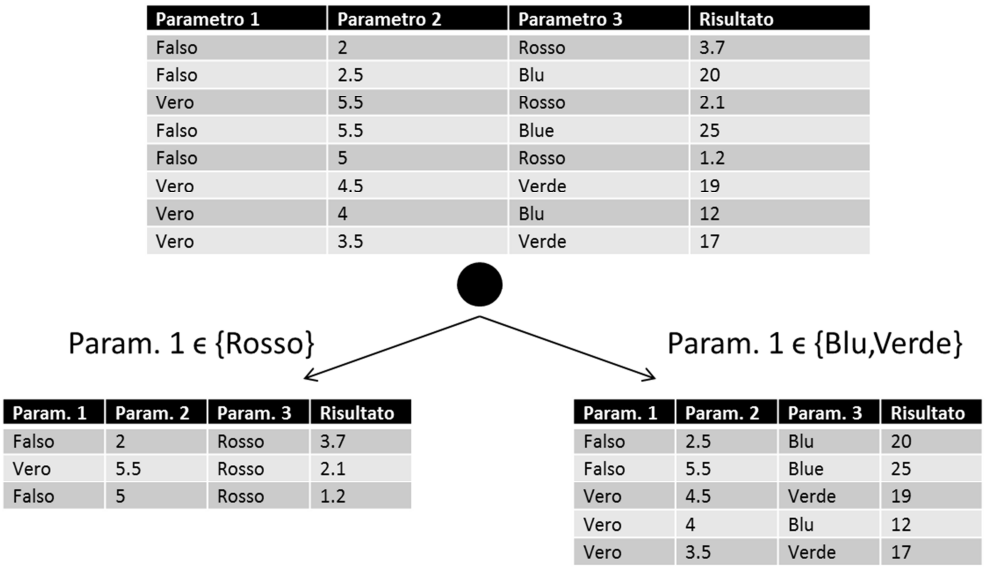
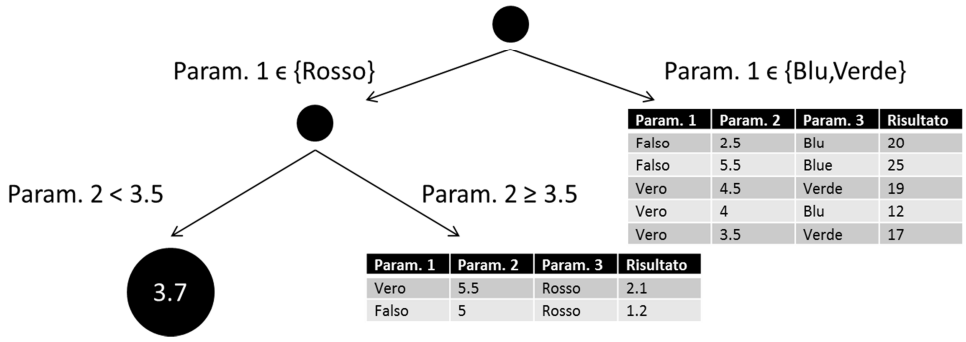


Figura 6 Regression tree d'esempio dopo primo split

Parametro 1	Parametro 2	Parametro 3	Risultato
Falso	2	Rosso	3.7
Falso	2.5	Blu	20
Vero	5.5	Rosso	2.1
Falso	5.5	Blue	25
Falso	5	Rosso	1.2
Vero	4.5	Verde	19
Vero	4	Blu	12
Vero	3.5	Verde	17



Param. 1	Param. 2	Param. 3	Risultato
Falso	2.5	Blu	20
Falso	5.5	Blue	25
Vero	4.5	Verde	19
Vero	4	Blu	12
Vero	3.5	Verde	17

Param. 1	Param. 2	Param. 3	Risultato
Vero	5.5	Rosso	2.1
Falso	5	Rosso	1.2

Figura 7 Regression tree d'esempio dopo secondo split

3.2 Random Forest

Il modello random forest utilizzato da SMAC è uno strumento utilizzato in machine learning per regressione e classificazione. Una Random Forest è composta da un insieme di alberi di regressione (da cui il termine foresta), ognuno dei quali viene costruito a partire da un insieme di dati d'ingresso campionati a caso dall'intero data set. Ad ogni punto di split di ogni albero solo un sottoinsieme, scelto a caso, dei parametri viene considerato eleggibile per essere utilizzato. Il risultato della Random Forest è ottenuto usando una funzione di aggregazione sui risultati dei singoli alberi (moda, media, ecc.).

I Regression Tree sono noti per avere buone prestazioni nel caso di parametri categorici e questa caratteristica viene ereditata da Random Forest, quest'ultimi, inoltre, dato la loro caratteristica di aggregare più alberi insieme sono più performanti di un albero singolo.

4 Fase sperimentale

Si vuole analizzare come ridurre il numero di bit utilizzati dalle variabili a virgola mobile mantenendo l'errore ad un livello prefissato.

Per poter analizzare il comportamento dell'applicazione modificando il tipo di dati utilizzato, è stato necessario riscriverne il codice sostanzialmente, utilizzando la libreria GNU MPFR (17). Questa libreria, che serve per il calcolo dei numeri in virgola mobile con arrotondamento corretto, permette di indicare quanti bit di mantissa usare per ogni variabile del programma. Utilizzando questo sistema è possibile impostare una qualsiasi variabile come se fosse un variabile a virgola mobile a 32, 16 e 8 bit; la selezione del tipo di variabile passa per un file di configurazione che contiene per ogni variabile il numero richiesto di bit per la mantissa. Il file di configurazione è letto a run-time dal programma, per cui non è necessaria una compilazione per ogni configurazione diversa. Questo file consiste in una serie di numeri separati da virgola per i quali ogni valore indica i bit di mantissa di una variabile, l'ordine in cui si trovano è lo stesso che hanno all'interno del programma (prima sono presenti le variabili della PCA, poi quella di DWT e infine quelle SVM). Questo sistema permette di controllare ogni variabile del programma indipendentemente, ma per gli scopi di questa tesi è necessario un controllo meno fine, di conseguenza è stato necessario isolare le variabili di ogni kernel. Ad ogni variabile corrisponde il componente a cui fanno riferimento. È stato usato uno script in associazione con il parser pycparser (18), in modo da estrarre l'Abstract Syntax Tree dal programma e poi ottenere gli identificatori

delle variabili e la funzione a cui appartengono. Una volta unite queste informazioni, è stato possibile costruire una mappa che lega ad ogni posizione del file di configurazione il kernel a cui si riferisce. Così, al fine di impostare l'applicazione per utilizzare, ad esempio, la DWT con solo variabili a 16 bit, è sufficiente impostare le variabili appartenenti a questa funzione come float16. Con questo processo è sufficiente invocare uno script con i corretti parametri ed indicare che tipo di configurazione si vuole analizzare per poter generare il file di configurazione corretto.

In preparazione all'utilizzo di SMAC sono stati usati i dati di 14'400 campionamenti in modo da creare un database che contenesse tutte le 2'187 configurazioni per ogni finestra di dati, per evitare di rieseguire qualche combinazione già vagliata.

Alla luce del fatto che SMAC può ottimizzare una sola metrica per volta, senza integrarne una di costo, è stato necessario trovare una soluzione che permettesse di considerare non solo il numero di bit totale dell'applicazione da minimizzare, ma anche un costo da rispettare, che in questo caso viene espresso in forma di vincolo. Il vincolo in questione consiste nel richiedere che il risultato finale non si discosti più di un certo range dal gold standard, cioè il risultato dell'applicazione utilizzando tutte variabili a virgola mobile a 64 bit. Per risultato dell'applicazione si prende una delle ultime variabili a virgola mobile all'interno del componente SVM, in modo che il risultato analizzato subisca ripercussioni da tutti i kernel dell'applicazione.

Le configurazioni che non soddisfano il vincolo restituiscono, a SMAC, un valore molto grande, che sta ad indicare all'algoritmo che sono soluzioni non soddisfacenti.

4.1 Impostazione del problema in SMAC

Per utilizzare SMAC è necessario definire uno scenario. Nel file scenario sono presenti tutte le informazioni necessarie per eseguire l'algoritmo in questione.

```
use-instances = true
runObj = QUALITY
numberOfRunsLimit = 3000
pcs-file = example_scenarios/seizure2/params.pcs
instance_file = example_scenarios/seizure2/instances-train.txt
test_instance_file = example_scenarios/seizure2/instances-
test.txt
algo = example_scenarios/seizure2/seizure.py
```

In questo caso viene indicata questa serie di condizioni:

- utilizzo di istanze:
- ottimizzazione del risultato ottenuto tramite algoritmo (non del tempo necessario per l'esecuzione)
- numero massimo di esecuzioni dell'algoritmo,
- file che contengono i parametri da ottimizzare,
- file e che contengono le istanze di test e di training
- file da chiamare per eseguire l'algoritmo.

I parametri vengono così definiti nel file param.pcs

```
mc ordinal {0,1,2} [0]
hh ordinal {0,1,2} [0]
ac ordinal {0,1,2} [0]
dg ordinal {0,1,2} [0]
pc ordinal {0,1,2} [0]
wv ordinal {0,1,2} [0]
sv ordinal {0,1,2} [0]
```

Esiste un parametro per ogni componente, ognuno definito come un ordinale con tre possibili valori, con lo zero come default. Lo zero indica variabili a 32 bit, l'uno a 16 e il due, di conseguenza, a 8.

Una volta avviato SMAC esso avvierà il file definito dall'algoritmo, fornendo come parametro l'istanza da utilizzare e la configurazione richiesta. L'algoritmo così chiamato può svolgere qualsiasi operazione, l'unica limitazione è che l'algoritmo chiamato deve scrivere su standard output una stringa del genere "Result of algorithm run: SUCCESS, 0, 0, 1000, 0" che indichi a SMAC che l'esecuzione è stata un successo (primo parametro) e la qualità della soluzione (quarto parametro), SMAC cercherà di minimizzare quest'ultimo. Esistono altri stati oltre al successo come l'ABORT e il CRASHED utilizzati per indicare a SMAC la presenza di qualche errore.

L'algoritmo utilizzato è il seguente:

```
#!/usr/bin/python
import sys, subprocess, os, sqlite3

#Imposta l'errore massimo al 10%
margin=10

left_margin=(100-margin)/100
rigth_margin=(100+margin)/100
```

```
#Lettura del file istanza da usare
if len(sys.argv)>1:
    instance = sys.argv[1]
else:
    print "Result of algorithm run: ABORT, 0, 0, 3, 0"

#Lettura della configurazione da usare  per ogni componente
#(0=32bit,1=16bit,2=8bit)
mc = 0
hh = 0
ac = 0
dg = 0
pc = 0
wv = 0
sv = 0

for i in range(len(sys.argv)-1):
    if (sys.argv[i] == '-mc'):
        mc = int(sys.argv[i+1])
    elif(sys.argv[i] == '-hh'):
        hh = int(sys.argv[i+1])
    elif(sys.argv[i] == '-ac'):
        ac = int(sys.argv[i+1])
    elif(sys.argv[i] == '-dg'):
        dg = int(sys.argv[i+1])
    elif(sys.argv[i] == '-pc'):
        pc = int(sys.argv[i+1])
    elif(sys.argv[i] == '-wv'):
        wv = int(sys.argv[i+1])
    elif(sys.argv[i] == '-sv'):
        sv = int(sys.argv[i+1])
os.chdir("/home/marco/smac-v2.10.03-master-
778/example_scenarios/seizure2/")
#Lettura da file del gold standard per ogni istanza
gold_values={}
with open('gold.txt') as f:
    for line in f:
        (key,value,stuff)= line.strip().split(',')
        gold_values[key]=float(value)
#Connessione al database contenente tutte le configurazioni già
analizzate
conn=sqlite3.connect('raw_data.db')
cursor=conn.cursor()
#Ricerca della configurazione richiesta all'interno del database
cursor.execute('SELECT result,bits FROM Raw_data01 WHERE
window=? AND mc=? AND hh=? AND ac=? AND dg=? AND  pc=? AND wv=?
AND sv=?;',(instance,mc,hh,ac,dg,pc,wv,sv))
query_result = cursor.fetchall()
print(query_result)
#Se la configurazione non è già stata eseguita viene svolta e
poi inserita nel database
if len(query_result)==0:
      cmd="./script_seizure.sh %s %d %d %d %d %d %d %d " % (
instance, mc, hh, ac, dg, pc, wv, sv )
```

```
      result=subprocess.check_output(cmd,stderr=subprocess.STDOU
T,shell=True)
      (value,bits,stuff)=result.split(',')
      print(result)
            try:
                  cursor.execute('INSERT INTO Raw_data01
VALUES(?, ?,?,?,?,?,?,?,
?,?)',(instance,mc,hh,ac,dg,pc,wv,sv,value,bits))
                  conn.commit()
            except:
                  print('Already done')
else:
    (value,bits)=query_result[0]
conn.close()
#Se la configurazione restuisce NotANumber deve essere scartata
if "nan" in value:
    print "Result of algorithm run: SUCCESS, 0, 0, 2000000000,
0"
else:
    value=float(value)
    ref_value=gold_values[instance]
#Se la configurazione rientra nel margine restituisce il numero
di bit utilizzati dalla stessa
    if value>ref_value*left_margin and
value<ref_value*right_margin:
        print "Result of algorithm run: SUCCESS, 0, 0, %d, 0" %
int(bits)
#Se la configurazione restuisce un numero fuori dal range deve
essere scartata
    else:
        print "Result of algorithm run: SUCCESS, 0, 0, %d, 0" %
1000000
```

L'algoritmo si comporta in tale modo: dopo aver ottenuto da SMAC quale istanza e quale configurazione eseguire, verifica se questa composizione è già all'interno del database di quelle eseguite, in caso positivo prosegue con l'esecuzione, altrimenti svolge il programma vero e proprio con la configurazione richiesta e inserisce il risultato nel database per riferimenti futuri. Una volta ottenuto il risultato si verifica che sia un valore valido (alcune configurazioni potrebbero causare stati inconsistenti restituendo come risultato NotANumber) e che si rispetti il vincolo sull'errore, in caso affermativo si restituisce il numero di bit

utilizzati da questa configurazione, altrimenti si restituisce un valore molto grande per rendere la configurazione non appetibile a SMAC.

4.2 Risultati

Per questi tre esperimenti sono state scelte le soglie d'errore pari a 10%, 30%, e 50%. È stato impostato il numero massimo d'esecuzioni dell'algoritmo pari a 200'000. Ogni prova è stata ripetuta cinque volte dato il non determinismo di SMAC.

I risultati degli esperimenti sono i seguenti:

- Nel caso con range al 10%, tutte e cinque le configurazioni risultanti sono quelle con tutte le variabili a 32 bit.
- Nel caso con range al 30%, tre configurazioni sono totalmente a 32 bit, mentre una ha impostato i tre kernel, mean covariance, householder e accumulate a 16 bit e il resto a 32. L'ultima configurazione risultante è totalmente a 32 bit ad eccezione del kernel accumulate che è stato utilizzato a 16 bit.
- Nel caso con range al 50%, le configurazioni trovate sono state 4 in cui solo il kernel accumulate fosse a 16 bit e il resto a 32 bit, e un ultima in cui tutti i kernel fossero 32.

La configurazione con il solo kernel accumulate a 16 bit ha un "consumo" di bit pari a 2'883 (vengono escluse tutte le variabili non a virgola mobile e quelle che non rientrano direttamente nelle operazioni dei kernel), corrispondente ad un risparmio del 3,9% percento rispetto al caso totalmente a 32 bit. L'altra configurazione trovata nell'altro caso invece porta ad un risparmio del 14,7%.

5 Confronto con un altro approccio

Parallelamente a questo lavoro, ne è stato seguito un altro, utilizzando Il progetto fpPrecisionTuning (19), con obiettivo analogo. Questo progetto è composto da due parti, C2mpfr e DistributedSearch. Il primo permette di convertire un applicazione C classica in una che utilizza la libreria MPFR. Questo tool è quello che è stato utilizzato per generare il codice sorgente in entrambi i casi di studio. Il secondo componente, invece, è quello responsabile di eseguire la ricerca della configurazione più soddisfacente, cercando di abbassare l'errore sotto una certa soglia, per cui è analogo all' utilizzo di SMAC fatto in questo lavoro.

La DistributedSearch utilizza un algoritmo definito come "distributed heuristic precision tuning". L'algoritmo consiste in due passi principali, entrambi parallelizzabili con approccio a scambio di messaggi (MPI (20)).

Inizialmente si esegue una binary search ripetuta su ogni singola variabile, fino a trovare la precisione minore consentita per quella variabile, nell'ipotesi di lasciare invariate le altre. Il risultato sono quindi n variabili a virgola mobile nel programma, la binary search viene eseguita n volte in parallelo. I singoli risultati vengono inseriti in un singolo vettore e consegnati al passo successivo.

Il secondo passo è un incremento della precisione eseguito su ogni variabile in parallelo, ripetuto iterativamente fino a rispettare i vincoli sull'errore in uscita. Ad ogni passo del ciclo, il numero di bit nella mantissa della variabile in esame e di quelle da essa dipendenti viene

incrementato di uno (una variabile dipende da un'altra se una qualsiasi espressione per il calcolo della prima richiede la seconda). Al termine del secondo passo, si seleziona tra le n configurazioni risultanti quella con il minore numero di bit.

I due passi proseguono fino a quando l'esecuzione del secondo passo non apporta più modifiche al risultato del primo. Questa condizione corrisponde ad uno minimo nello spazio degli stati, in quanto non è più possibile diminuire né aumentare il numero di bit.

Per avere dei risultati confrontabili gli esperimenti sono stati svolti in questo modo: per ognuno degli otto campioni è stata svolta una ricerca per cui l'errore fosse pari al 10%. Il tempo d'esecuzione richiesto da DistributedSearch è stato dato come tetto massimo per l'esecuzione di SMAC. Inizialmente le analisi sono state svolte solo per un campione per volta.

Visto che DistributedSearch ha una granularità fine al singolo bit per la mantissa, per confrontare i risultati con quelli di SMAC, è necessario scalare i valori al più piccolo tipo di dati (float32, float16, ecc.) che lo possa contenere, per cui, ad esempio, una variabile che utilizzi cinque bit di mantissa viene scalata come un float16. È possibile fare questa considerazione perché, aumentando il numero di bit della mantissa, la precisione non dovrebbe peggiorare.

Nella Tabella 3 sono indicate, per ogni componente, il numero di variabili di ogni tipo che sono state ritenute utilizzabili, per entrambi gli approcci.

Campione	DistributedSearch							SMAC							#bit
	MC	HH	AC	DG	PC	DWT	SVM	MC	HH	AC	DG	PC	DWT	SVM	
0	0	0	0	0	1	0	1	0	0	0	0	5	0	19	**32**
	2	1	1	7	1	2	11	5	0	0	41	0	13	0	**16**
	3	19	8	34	3	11	7	0	20	9	0	0	0	0	**8**
1	0	0	0	0	1	0	1	0	0	0	0	0	0	19	**32**
	2	1	1	8	1	2	11	5	0	0	41	0	0	0	**16**
	3	19	8	33	3	11	7	0	20	9	0	5	13	0	**8**
2	0	0	0	0	1	1	1	0	0	0	0	5	13	19	**32**
	4	17	9	27	4	9	12	5	0	0	41	0	0	0	**16**
	1	3	0	14	0	3	6	0	20	9	0	0	0	0	**8**
3	0	0	0	0	1	1	1	0	0	0	0	0	0	19	**32**
	2	4	0	5	1	1	11	1	0	0	0	0	0	0	**16**
	3	16	9	36	3	11	7	4	20	9	41	5	13	0	**8**
4	0	0	0	0	1	1	1	0	0	0	0	0	0	19	**32**
	1	0	1	7	1	3	10	0	0	0	41	0	0	0	**16**
	4	20	8	34	3	9	8	5	20	9	0	5	13	0	**8**
5	0	0	0	0	1	1	1	0	0	0	0	0	0	19	**32**
	1	2	4	9	1	4	10	0	0	0	41	0	13	0	**16**
	4	18	5	32	3	8	8	5	20	9	0	5	0	0	**8**
6	0	0	0	0	1	1	1	0	0	9	0	5	13	19	**32**
	2	7	8	15	4	9	10	5	20	0	41	0	0	0	**16**
	3	13	1	26	0	3	8	0	0	0	0	0	0	0	**8**
7	0	0	0	0	0	0	1	0	0	0	0	0	0	19	**32**
	0	0	0	41	0	3	11	0	0	0	41	0	0	0	**16**
	5	20	9	0	5	10	7	5	20	9	0	5	13	0	**8**

Tabella 3 Confronto risultati tra DistributedSearch e SMAC

SMAC può ottimizzare solo con la granularità del singolo kernel per cui, tutte le variabili di un singolo componenti condividono lo stesso tipo di dati, cosa che nell'altro approccio non è vera.

Da questi esperimenti si evidenzia che l'utilizzo di DistributedSearch permette un risparmio medio del 45% rispetto alle soluzioni trovate da SMAC.

DistributedSearch di base è in grado di ottimizzare una sola istanza per volta, quindi per svolgere un test è stato necessario svolgere alcune modifiche al software. Provando ad usare questa versione sperimentale, dopo diverso tempo, si converge verso una configurazione, ma l'algoritmo non è in grado di concludersi correttamente.

DistributedSearch							SMAC							#bit
MC	HH	AC	DG	PC	DWT	SVM	MC	HH	AC	DG	PC	DWT	SVM	
0	0	0	1	1	1	3	0	0	9	0	5	13	19	**32**
4	19	9	29	4	11	10	5	20	0	41	0	0	0	**16**
1	1	0	11	0	1	6	0	0	0	0	0	0	0	**8**

Tabella 4 Confronto risultati tra DistributedSearch e SMAC in configurazione multi-istanza

Anche questa versione, seppur preliminare, è in grado di trovare una soluzione più performante di SMAC con un risparmio di bit paragonabile a quello presente nella versione per singola istanza.

6 Conclusioni

Il Transprecision Computing rappresenta la prossima frontiera per lo stoccaggio, la comunicazione e l'elaborazione di grandi quantità di dati. Il progetto OPRECOMP, in quest'ottica, mira a costruire un vero e proprio nuovo paradigma, potenzialmente utile per moltissime applicazioni. Al fine di raggiungere questo obiettivo ambizioso, è fondamentale testare nuove strade e strumenti. Lo scopo di questa tesi consisteva nell'operare su strumenti nuovi e per questo, perfezionabili. Sebbene non tutte le aspettative su questo lavoro siano state soddisfatte, l'utilizzo di sistemi di configurazioni automatici si è rilevato in grado di fare quanto predetto, minimizzare il numero di bit necessari per l'applicazione in questione. Inoltre si è visto che l'approccio utilizzato con SMAC è generalizzabile e di facile utilizzo. La conversione all'utilizzo della libreria MPFR è necessaria per ogni approccio del genere e non presenta particolari difficolta. Esistono inoltre alcuni punti a sfavore di SMAC, come l'impossibilità di eseguire una ricerca in parallelo e l'obbligo di utilizzare una sola funzione obiettivo, costringendo all'utilizzo di work-around come quello utilizzato impostando il vincolo. Questi problemi non sono presenti nel caso di DistributedSearch, visto che è stato realizzato in principio come applicazione parallela. Inoltre quest'ultimo ha un controllo più fine delle variabili, ma non utilizza nessun tipo di modello e ha solo un supporto parziale per l'ottimizzazione di più campioni contemporaneamente.

Riferimenti

1. *iACT: A Software-Hardware Framework for Understanding the Scope of Approximate Computing.* **Asit K. Mishra, Rajkishore Barik and Somnath Paul.**

2. *Reducing power by optimizing the necessary precision/range of floating-point arithmetic.* **J.Y.F. Tong, D. Nagle and R.A. Rutenbar.**

3. *A Taxonomy of Approximante Computing Tecniques.* **Thierry Moreau, Joshua San Miguel, Mark Wyse, James Bornholt, Luis Ceze, Natalie Enright Jerger and Adrian Sampson.**

4. http://grouper.ieee.org/groups/754/. [Online]

5. https://upload.wikimedia.org/wikipedia/commons/d/d2/Float_example.svg. [Online]

6. *Minimizing Floating-Point Power Dissipation Via Bit-Width Reduction.* **Ying Fai Tong, Rob A. Rutenbar, and David F. Nagle.**

7. http://oprecomp.eu/. [Online]

8. *Application of machine learning to epileptic seizure onset detection and treatment.* **Shoeb, Ali Hossam.**

9. https://en.wikipedia.org/wiki/Principal_component_analysis. [Online]

10. https://it.wikipedia.org/wiki/Wavelet. [Online]

11. https://en.wikipedia.org/wiki/Discrete_wavelet_transform. [Online]

12. https://it.wikipedia.org/wiki/Macchine_a_vettori_di_supporto. [Online]

13. **Lin, Chih-Chung Chang and Chih-Jen.** LIBSVM . [Online] http://www.csie.ntu.edu.tw/~cjlin/libsvm.

14. **M. Galassi et al.** *GNU Scientific Library Reference Manual (3rd Ed.), ISBN 0954612078.*

15. *Sequential Model-Based optimization for General Algorithm Configuration.* **Frank Hutter, Holger H. Hoos and Kevin Leyton Brown.**

16. *Sequential Model-based Optimization.* **Frank Hutter, Holger Hoos, Kevin Leyton-Brown.** Roma : s.n., 2011.

17. **Laurent Fousse, Guillaume Hanrot,Vincent Lefèvre,Patrick Pélissier and Paul Zimmermann.** *MPFR: A multiple-precision binary floating-point library with correct rounding.*

18. https://github.com/eliben/pycparser. [Online]

19. https://github.com/minhhn2910/fpPrecisionTuning. [Online]

20. http://www.mpi-forum.org. [Online]