COME OTTIMIZZARE IN LABVIEW APPLICAZIONI DI TEST AUTOMATIZZATI PER PROCESSORI MULTICORE

David Hall

Vediamo come delle applicazioni scritte in LabVIEW possono essere ottimizzate sfruttando tecniche di programmazione parallela

LabVIEW offre un ambiente di programmazione grafica unico e di semplice utilizzo per le applicazioni di test automatizzati, ma è la sua capacità di assegnare dinamicamente parti di codice ai vari core della CPU ad aumentare la velocità di esecuzione sui processori multicore.

LA SFIDA DELLA PROGRAMMAZIONE MULTITHREADED

Fino a poco tempo fa, le innovazioni apportate nella tecnologia dei processori si traducevano in computer con CPU (Central Processing Unit) in grado di funzionare a frequenze di clock più elevate. Tuttavia, con l'avvicinarsi di tali frequenze al loro limite fisico teorico, sono stati sviluppati nuovi processori con più core di elaborazione, anziché uno solo. Con i nuovi processori multicore, le applicazioni di test automatizzati raggiungono le migliori prestazioni e i volumi di lavoro più elevati quando si utilizzano tecniche di programmazione parallela. Edward Lee, professore di Ingegneria Elettrotecnica ed Informatica presso l'Università della California (Berkeley), descrive così i vantaggi dell'elaborazione parallela:

"Molti esperti di settore prevedono che la risposta ai limiti della Legge di Moore si darà sempre più con architetture di calcolo parallele. Per sperare di continuare ad ottenere prestazioni sempre migliori in questo campo, i programmi dovranno essere in grado di sfruttare al meglio questo parallelismo".

Programmare applicazioni in grado di sfruttare tecnologie hardware parallele rappresenta una grande sfida in campo informatico.

Fortunatamente, LabVIEW offre un ambiente di programmazione ideale per i processori multicore, perché offre un ambiente intuitivo per creare algoritmi paralleli e può assegnare dinamicamente thread multipli a una data applicazione. Infatti, le applicazioni di test automatizzati che utilizzano processori multicore possono essere facilmente ottimizzate per raggiungere le migliori prestazioni. Grandi benefici ne possono ricavare in particolare gli strumenti modulari PXI Express, ad esempio in applicazioni di analisi di segnale multicanale e di elaborazione Hardware in the Loop, grazie alle velocità elevate di trasferimento dati ottenibili con il bus PXI Express. Nel seguito, valuteremo varie tecniche di programmazione parallela e caratterizzeremo in termini di prestazioni i benefici offerti da ciascuna tecnica.

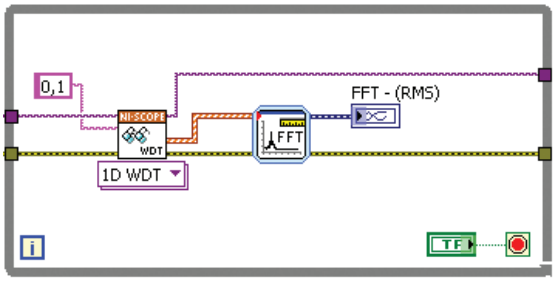

Figure 1 – Codice LabVIEW per l'esecuzione sequenziale

IMPLEMENTAZIONE DI ALGORITMI DI TEST PARALLELI

Una comune applicazione di test automatizzati (ATE) che trae vantaggio dall'elaborazione parallela è l'analisi di segnale multicanale. Poiché l'analisi in frequenza è normalmente un'operazione onerosa per i processori, la velocità di esecuzione può essere migliorata parallelizzando il codice di test, in modo tale che l'elaborazione di segnale di ogni canale possa essere distribuita su più core di elaborazione. Dal punto di vista del programmatore, l'unica modifica richiesta per ottenere questo beneficio è semplicemente quella di ristrutturare l'algoritmo di test.

Per illustrare il concetto, confronteremo i tempi di esecuzione di due algoritmi di analisi in frequenza multicanale operanti su due canali di un digitalizzatore ad alta velocità. Nel nostro test viene impiegato un digitizer a 14 bit PXIe-5122 per acquisire i segnali alla massima velocità di campionamento consentita (100 MS/s). Innanzitutto, vediamo in LabVIEW il modello di programmazione

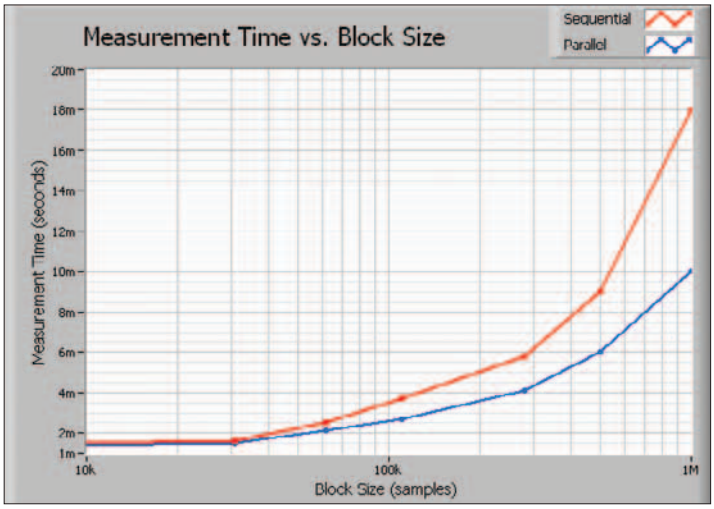

Figura 3 – Tempo di esecuzione degli algoritmi sequenziale e parallelo

sequenziale tradizionale necessario per effettuare questa operazione.

Nello schema a blocchi della fig. 1, l'analisi in frequenza di entrambi i canali viene eseguita in un VI Express per misure spettrali basate su FFT che analizza ciascun canale in serie. Anche se l'algoritmo della fig. 1 può essere comunque eseguito efficientemente su processori multicore, è possibile migliorarne le prestazioni elaborando ciascun canale in parallelo.

Se analizzassimo i dettagli esecutivi dell'algoritmo precedente, noteremmo che la FFT richiede un tempo significativamente più lungo per essere completata rispetto all'acquisizione dal digitalizzatore ad alta velocità. Acquisendo separatamente i due canali ed eseguendo due FFT in parallelo, possiamo ridurre notevolmente il tempo di elaborazione. Utilizzando l'approccio parallelo, si ottiene il nuovo schema a blocchi illustrato nella fig. 2.

I canali del digitalizzatore vengono coinvolti sequenzialmente. Notate che queste operazioni potrebbero essere eseguite completamente in parallelo se le due operazioni di fetch fossero associate a strumenti distinti. Tuttavia, dato che sono le operazioni di trasformazione ad impegnare particolarmente il processore, riusciamo ancora a migliorare le prestazioni semplicemente parallelizzando l'elaborazione di segnale. Di conseguenza, il tempo di esecuzione totale si riduce. Nella fig. 3 è illustrato il tempo di esecuzione delle due implementazioni.

All'aumentare delle dimensioni del blocco (campioni per fetch), il tempo di elaborazione risparmiato attraverso l'esecuzione parallela diventa molto più evidente. Infatti, l'algoritmo parallelo si avvicina a un raddoppio delle prestazioni per i blocchi di dimensioni più grosse. Il grafico nella fig. 4 illustra l'esatto aumento percentuale di prestazioni in funzione delle dimensioni dell'acquisizione (in campioni).

Per dimensioni dei blocchi maggiori di 1 milione di campioni (ampiezza di banda di risoluzione pari a 100 Hz), l'approccio parallelo si traduce in aumenti di prestazioni di almeno l'80%.

Ottenere un aumento di prestazioni delle applicazioni di test automatizzati sui processori multicore è facile in LabVIEW, perché l'ambiente alloca dinamicamente ogni thread. Infatti, agli utenti non è richiesto di creare codice speciale per abilitare il multithreading, mentre le applicazioni di test parallelo possono trarre beneficio dai processori multicore con minimi aggiustamenti della programmazione.

CONFIGURAZIONE DEGLI ALGORITMI DI TEST PARALLELO CUSTOMIZZATO

Il beneficio della parallelizzazione degli algoritmi di elaborazione dei segnali è che permette a LabVIEW di suddividere l'uso della CPU fra più core.

LabVIEW è in grado di elaborare in parallelo gran parte dei dati acquisiti, risparmiando tempo di esecuzione.

Un requisito per l'elaborazione parallela è che LabVIEW

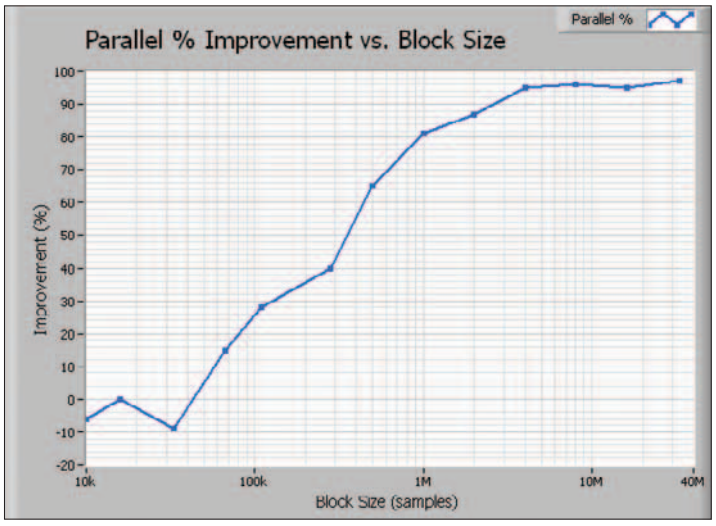

Figura 4 – Aumento di prestazioni degli algoritmi paralleli (in percentuale)

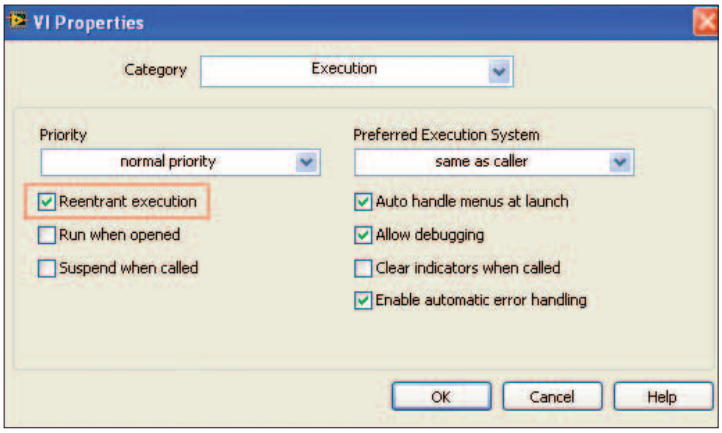

Figura 5 – Configurazione per l'esecuzione rientrante in LabVIEW

esegua una copia (o clone) di ciascuna subroutine di elaborazione del segnale. Di default, molti degli algoritmi di elaborazione dei segnali di LabVIEW sono configurati per garantire un'esecuzione rientrante. Ciò significa che LabVIEW alloca dinamicamente una singola istanza di ciascuna subroutine – inclusi thread distinti e spazio di memoria. Di conseguenza, le subroutine custom devono essere configurate affinché funzionino in modo rientrante. Ciò si può fare con un semplice step di configurazione in LabVIEW. Per impostare questa proprietà, selezionate File >> VI Properties e scegliete la categoria "Execution". Selezionate quindi il flag che abilita l'esecuzione rientrante come illustrato nella fig. 5.

Con il semplice passo mostrato in figura è possibile eseguire in parallelo più subroutine custom, come accade per le funzioni di analisi standard presenti in LabVIEW. Di conseguenza, le applicazioni di test automatizzati possono raggiungere migliori prestazioni sui processori multicore tramite semplici tecniche di programmazione.

OTTIMIZZAZIONE DELLE APPLICAZIONI HARDWARE-IN-THE-LOOP

Un altro esempio di applicazioni che possono trarre vantaggio dalle tecniche di elaborazione parallela del segnale, utilizzando più strumenti per gestire input e output simultanei, sono le applicazioni hardware-in-the loop (HIL) e di elaborazione in linea. Prendiamo ad esempio il caso di un'acquisizione che utilizzi un digitalizzatore ad alta velocità o un modulo di I/O digitale ad alta velocità. A livello software, viene eseguito un algoritmo di elaborazione digitale del segnale. Infine, il risultato viene generato da un altro strumento modulare. Nella fig. 6 è illustrato un tipico schema a blocchi.

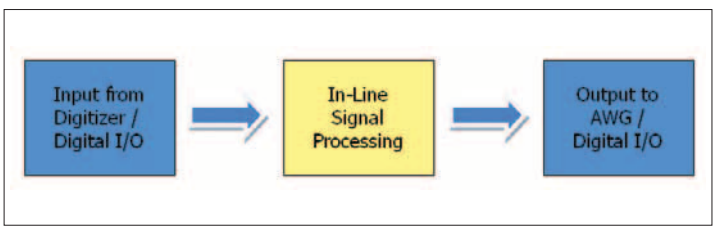

Figura 6 – Schema a blocchi dell'elaborazione di segnale in linea

Comuni applicazioni HIL includono la simulazione di sensori e l'emulazione di componenti custom. In questo articolo, esploreremo alcune tecniche per ottenere il migliore throughput per applicazioni in linea di elaborazione digitale dei segnali. In generale, si possono usare due strutture di programmazione fondamentali: la struttura a loop singolo e la struttura multiloop basata su pipeline con utilizzo di code. La struttura a loop singolo è semplice da implementare e presenta una bassa latenza per i blocchi di piccole dimensioni. Al contrario, le architetture multiloop possono raggiungere un throughput molto più elevato, perché sono in grado di utilizzare meglio le CPU multicore.

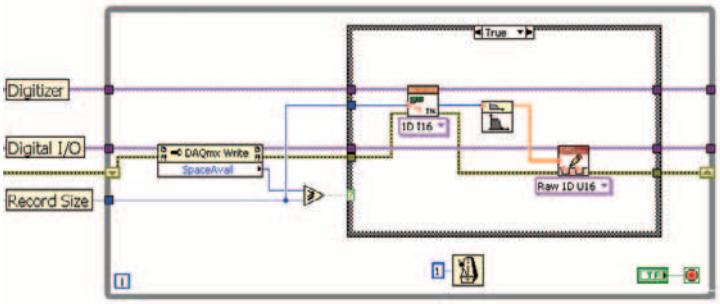

Figura 7 – Approccio a loop singolo al Processing in the Loop

Usando l'approccio tradizionale a loop singolo, mettiamo in ordine sequenziale una funzione di lettura di un digitalizzatore ad alta velocità, l'algoritmo di elaborazione del segnale e una scrittura di I/O digitale ad alta velocità. Come lo schema a blocchi nella fig. 7 illustra, ciascuna di queste subroutine deve essere eseguita in serie, secondo quanto determinato dal modello di programmazione a flusso di dati di LabVIEW.

La struttura a loop singolo è soggetta a diversi limiti. Poiché ogni stadio viene eseguito in serie, il processore è limitato nell'eseguire I/O dallo strumento mentre elabora i dati. Con questo approccio, una

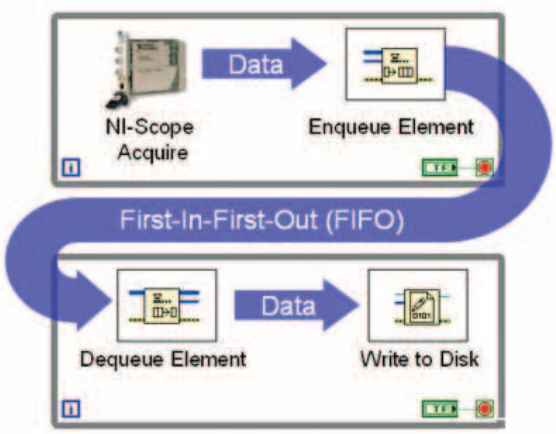

Figura 8 – Le code permettono la condivisione dei dati fra più loop

CPU multicore non può essere utilizzata in modo efficiente perché il processore può eseguire una sola funzione per volta. Quindi, verrà utilizzato un unico core di una CPU multicore per questa applicazione. Benché la struttura a loop singolo sia sufficiente per le velocità di acquisizione più basse, per gestire un throughput dati più elevato è richiesto un approccio multiloop.

L'architettura multiloop utilizza le code per passare i dati fra i vari cicli while. Nella fig. 8 illustriamo il concetto della programmazione a flusso di dati nel caso di più cicli while con l'utilizzo delle code. Le code permettono la condivisione dei dati fra più loop. La figura rappresenta quella che viene tipicamente chiamata 'struttura a ciclo produttore-consumatore'. In questo caso, un digitalizzatore ad alta velocità acquisice i dati in un primo loop e passa ad ogni iterazione un nuovo set di dati alla FIFO.

Il loop consumer monitorizza semplicemente lo stato della coda e scrive ogni nuovo set di dati sul disco quando diventa disponibile. Il valore dell'uso delle code è che entrambi i loop possono essere eseguiti indipendentemente fra loro. Nell'esempio precedente, il digitalizzatore ad alta velocità può continuare ad acquisire dati anche se c'è un ritardo nella loro scrittura su disco. Nel frattempo, i campioni in più vengono semplicemente memorizzati nella FIFO.

Generalmente, l'approccio produttore-consumatore a pipeline permette un throughput dati maggiore, consentendo un utilizzo più efficiente dei processori. Questo vantaggio è ancora più evidente nei processori multicore, perché LabVIEW può assegnare dinamicamente i thread della CPU ad ogni core.

Per un'applicazione di elaborazione del segnale in linea, possiamo usare tre while loop indipendenti e due code per passare i dati fra loro. In questo scenario, un loop acquisirà i dati da uno strumento, uno sarà dedicato all'elaborazione del segnale ed il terzo scriverà i dati su un secondo strumento. Nella fig. 9 è riportato uno schema a blocchi di LabVIEW che illustra questo approccio.

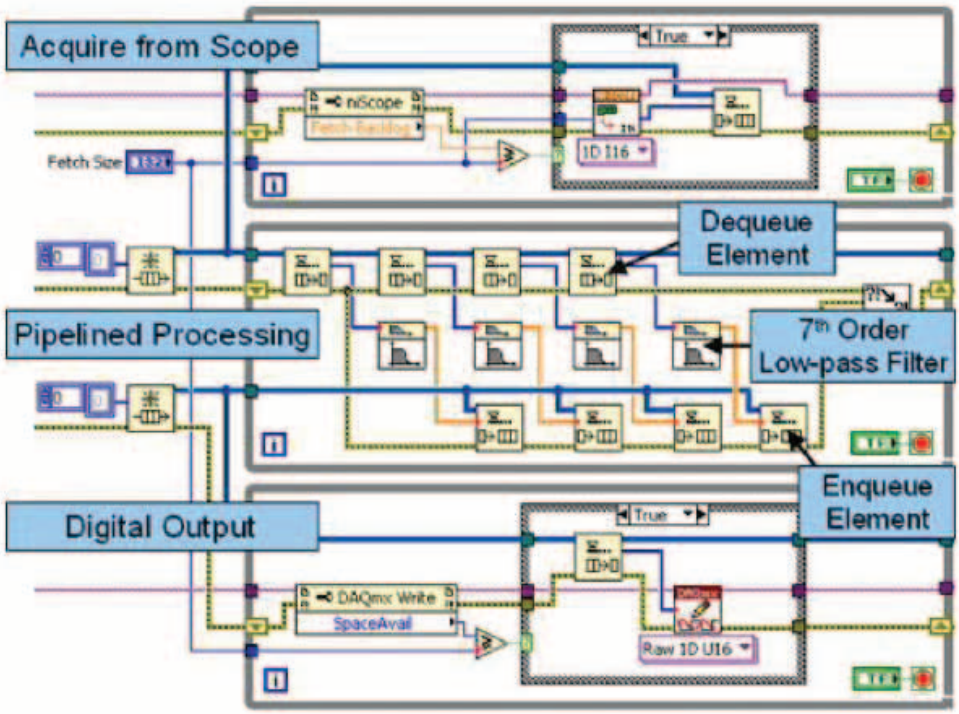

Figura 9 – Elaborazione del segnale a pipeline con loop multipi e code

Nella fig. 9, il loop superiore è un loop produttore che acquisisce dati da un digitalizzatore ad alta velocità e li passa alla prima struttura a coda (FIFO). Il loop intermedio opera sia come produttore che consumatore.

Ad ogni iterazione, il ciclo carica (consuma) diversi set di dati dalla prima coda e li elabora indipendentemente come pipeline. Questo approccio migliora le prestazioni di elaborazione nei processori multicore permettendo di elaborare indipendentemente fino a quattro set di dati. Notate che il loop intermedio opera anche come produttore, passando i dati elaborati nella seconda coda. Infine, il loop inferiore scrive i dati elaborati sul modulo di I/O digitale ad alta velocità.

Gli algoritmi di elaborazione parallela sfruttano il processore in modo più efficiente sulle CPU multicore. Infatti, il throughput totale dipende da due fattori: l'utilizzo del processore e le velocità di trasferimento sul bus. In generale,

la CPU e il bus dati funzionano in modo più efficiente quando si elaborano grossi blocchi di dati. Inoltre, possiamo ridurre ancora di più i tempi di trasferimento dei dati usando strumenti PXI Express, che hanno tempi di trasferimento più veloci. Di conseguenza, possiamo illustrare il massimo throughput in termini di velocità di campionamento in funzione dell'acquisizione espressa in numero di campioni, come si vede nella fig. 10.

Tutti i benchmark illustrati in questo grafico sono stati eseguiti su campioni a 16 bit. Inoltre, l'algoritmo di elaborazione del segnale usato era un filtro passa-basso Butterworth del 7° ordine con un cutoff di 0,45 moltiplicato per la velocità di campionamento. Come i dati illustrano, l'approccio a pipeline (multiloop) a 4 stadi permette di ottenere il throughput dati più elevato. Notate che un approccio di elaborazione del segnale a 2 stadi permette di ottenere prestazioni migliori del metodo a singolo loop (sequenziale), ma non utilizza la CPU con altrettanta efficienza del metodo a 4 stadi. Le velocità di campionamento elencate nelle tabelle 1 e 2 sono la massima velocità di campionamento di input e output per un digitalizzatore ad alta velocità PXIe-5122 ed un modulo di I/O digitale ad alta velocità PXIe-6537. Notate che a 20 MS/s, il bus trasferisce dati alle velocità di 40 MB/s per l'input e 40 MB/s per l'output, per un'ampiezza di banda totale del bus di 80 MB/s.

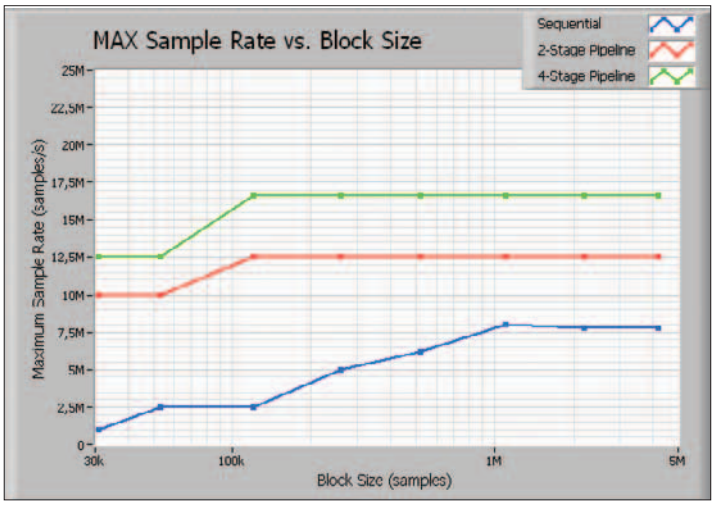

Figura 10 – Throughput delle strutture multiloop e a loop singolo

E' anche importante considerare che l'approccio di elaborazione a pipeline introduce latenza fra input e output.

La latenza dipende da diversi fattori, incluse le dimensioni dei blocchi e la velocità di campionamento. Le tabelle 1 e 2 confrontano la latenza misurata in funzione delle dimensioni dei blocchi e della massima velocità di campionamento per le architetture a loop singolo e multiloop a 4 stadi.

Come ci si poteva aspettare, la latenza aumenta mano a mano che l'utilizzo della CPU si avvicina al 100%. Ciò è particolarmente evidente nell'esempio della pipeline a 4 stadi, con velocità di campionamento di 20 MS/s. Al contrario, l'utilizzo della CPU supera di poco il 50% in tutti gli esempi a loop singolo.

TABELLA 1

Dimensioni del blocco	Velocità di campionamento (max)	Latenza
32k	1 MS/s	2,50 ms
64k	2,5 MS/s	5,62 ms
128k	2,5 MS/s	11,56 ms
256k	5 MS/s	22,03 ms
512k	6,25 MS/s	44,22 ms
1M	8,25 MS/s	85,63 ms
2M	8,28 MS/s	169,52 ms
4M	8,25 MS/s	199,62 ms

TABELLA 2

Dimensioni del blocco	Velocità di campionamento (max)	Latenza
32k	12,5 MS/s	38,78 ms
64k	12,5 MS/s	45,41ms
128k	16,67 MS/s	38,27 ms
256k	16,67 MS/s	44,86 ms
512k	16,67 MS/s	55,17 ms
1M	20 MS/s	148,85 ms
2M	20 MS/s	247,29 ms
4M	20 MS/s	581,15 ms

Tabella 1 e 2 – Benchmark di latenza per loop singolo e per pipeline a 4 stadi

CONCLUSIONE

La strumentazione basata su PC, come gli strumenti modulari PXI e PXI express, trae grandi benefici dai progressi della tecnologia dei processori multicore e dall'aumento della velocità dei bus dati. Mano a mano che le nuove CPU migliorano le prestazioni aggiungendo più core di elaborazione, sono necessarie strutture di elaborazione parallela o a pipeline per massimizzare l'efficienza della CPU. Fortunatamente, LabVIEW offre un'eccellente soluzione a questo problema di programmazione, assegnando dinamicamente i task di elaborazione ai singoli core di elaborazione. Come i dati sopra riportati evidenziano, si possono raggiungere significativi miglioramenti di prestazioni strutturando gli algoritmi di LabVIEW in modo da sfruttare l'elaborazione parallela.

Note sull'autore

David Hall, Signal Sources Product Engineer, National Instruments.