Accelerated Algorithmic Trading for Taiwan Future Market

適用台灣期貨市場之加速算法交易

組別:A211 指導教授:馬席彬教授 組員:108061127 許澤群

摘要

算法交易(Algorithmic Trading)係指運用程式快速和自動化的優勢，分析特定市場的數據後，根據交易策略的決策來獲取報酬。加速算法交易(Accelerated Algorithmic Trading)則是藉由硬體加速的算法交易。

在 Xilinx 所提供的開放資源中，存在一套適用於芝加哥期貨市場的加速算法交易模型，因此本專題之研究目的為以該模型為參考原型，修改或重新設計一套適用於台灣期貨市場的算法交易模型，並透過 Alveo U50數據中心加速卡進行加速。

專題流程從收集資料開始，首先必須認識網際網路資料傳輸原理以及交易所傳輸資訊的種類和格式，並分析 Xilinx 的加速算法交易的模組架構。之後設計解碼交易所封包和收集市場數據的程式，並以元大金融控股公司所錄製的封包驗證解碼結果。

研究結果完成了適用於台灣期貨市場的加速算法交易部分模型之軟體模擬與合成，硬體驗證和其他模型的設計則是後續的目標。

內容

一、前言

算法交易(Algorithmic Trading)係指運用電腦程式快速和自動化的優勢，分析特定市場的數據資料後，根據事先設定的交易策略，決定下單的時機、價格和數量後進行操作，並從中獲得報酬。而透過硬體提高算法交易之程式的運作速度，便形成加速算法交易(Accelerated Algorithmic Trading)。

在美國的可程式邏輯器件生產商 Xilinx 提供的開放資源中，包含了適用於芝加哥期貨市場的加速算法交易模型，目的在於降低交易策略開發人員實作算法交易的門檻。因此本專題之研究目的為以該加速算法交易模型為參考原型，修改或重新設計適用於台灣期貨市場的模型。

二、原理分析

算法交易的流程分為：接收並解碼來自交易所的封包資訊、根據資訊內容對商品委託簿進行變更、交易策略根據變更後的委託簿作出決策以及將決策結果包裝成委託封包。

第一步驟解碼台灣期貨交易所的封包，需參考期貨交易所釋出的逐筆行情資訊傳輸作業手冊，手冊記載各種類資料的格式以及說明。因為目標為取得委託簿變更的資訊，所以解碼對象以委託簿揭示訊息(I081)為主。第二步驟更新商品委託簿，同樣參考逐筆行情資訊傳輸作業手冊的說明來操作，操作種類主要分成新增、修改和刪除[1]。第三步驟作出決策，由預先設定好的交易策略根據最新委託簿五檔價量，作出送出、更新或取消買賣委託的決策。第四步驟由於缺少向交易所送出委託之封包的格式資料，因此現階段尚未完成。

另外，由於封包在進入網路傳輸之前會經過層層包裝，因此需先對封包進行拆封，取得封包的酬載後才能將其輸入算法交易的模組。拆封方式需參考各個協定所對應的檔頭格式，而 I081的協定(Ethernet、IPv4和 UDP)之檔頭格式皆為固定長度[2]，因此可以透過計算讀取的資料位元數目便可以得到檔頭資訊和封包的酬載。

三、系統設計

參考 Xilinx 的 AAT 模型後，根據上述算法交易的流程，將每一步驟用一個模組來實作，因此總共有三個模組：feedHandler、orderBook、pricingEngine。三個模組的關係圖如下[3]，

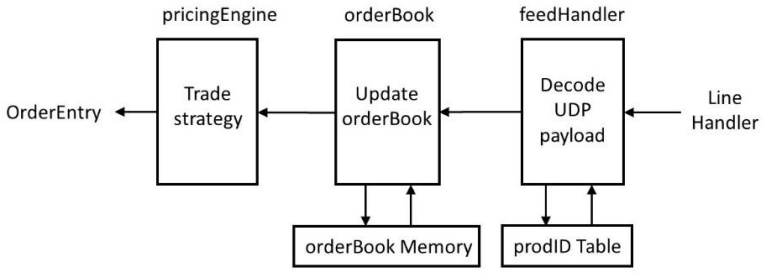

圖1 模組關係圖

feedHandler 負責解碼來自台灣期貨交易所的逐筆行情更新封包，orderBook 根據封包的內容更新對應商品的委託簿，pricingEngine 藉由委託簿五檔價量的變化來作出交易決策。

(一) feedHandler

feedHandler 模組的設計如下，

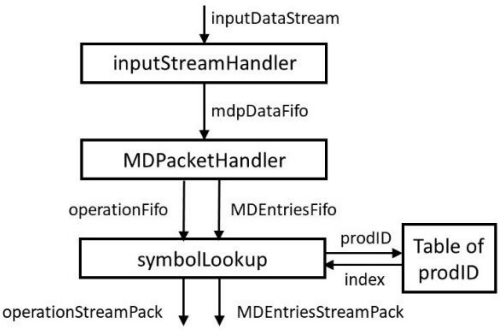

圖2 feedHandler 模組架構

inputStreamHandler 負責將輸入資料從數據流(data stream) 型態的轉換成64-bit 為一組的 FIFO 型態，該設計目的除了讓資料長度契合介面寬度，固定長度有助於定出當前資料在封包的對應位置。然而單一個 I081的 UDP 封包資料並非固定長度，也非64-bit 的整數倍，因此封包資料在進入該模組之前，需透過前處理將長度以補齊方式設置成64-bit 的倍數。補齊的位元組皆為0xff，因 I081封包資料當中不會出現此位元組，0xff 可用於辨識 I081封包的結束。

MDPacketHandler 接收64-bit 為一組的 FIFO 輸入資料後，根據逐筆行情資訊傳輸作業手冊中的 I081格式來解碼委託簿變更的相關資訊。I081的 UDP 封包內容分成三個部分：行情訊息共用檔頭、商品訊息以及委託簿變更條目[1]。有別於芝加哥期貨交易所之封包格式，單一 I081封包存在複數組委託簿變更條目資料，因此行情訊息共用檔頭和商品訊息的資料存取於同一資料類別，委託簿變更條目的資料則存取於另一資料類別，兩者皆以 FIFO 的方式傳輸給下一級的功能。

symbolLookup 接收解碼後的行情訊息共用檔頭、商品訊息和委託簿變更資料後，根據商品訊息中的商品代碼(prodID)，在商品代碼表中查詢對應的索引值(index)。若查詢成功則將該索引值和委託簿變更條目資料包裝成資料流的資料型態，並傳送給下一級的模組，同理行情訊息共用檔頭和商品訊息也做一樣的包裝；若查詢失敗則捨棄這筆委託簿變更資料。

(二) orderBook

orderBook 模組的設計如下，

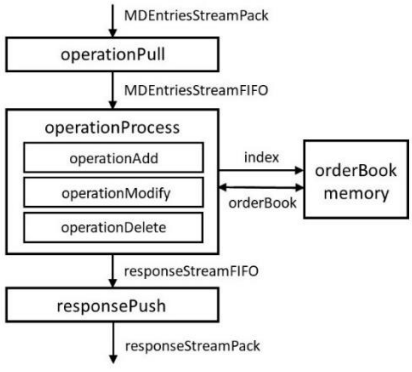

圖3 orderBook 模組架構

operationPull 接收包裝後的委託簿變更條目之數據流資料後，將輸入資料拆封成對應的資料類別，並以 FIFO 的方式傳輸給下一級的功能。

operationProcess 接收拆封後的委託簿變更資料後，根據商品代碼所對應的索引值(index)，在記憶體中取得該商品的委託簿五檔價量。接著根據委託簿變更的操作代碼，呼叫對應的功能來更新委託簿五檔價量，變更的操作分為四種：新增(Add)、修改(Modify)、刪除(Delete)和覆蓋(Overlay)。由於覆蓋和修改的更新方式雷同，兩種更新方式皆使用同一功能實現，因此共有三種功能來實現委託簿五檔價量的更新。最後更新後的委託簿五檔價量存取於對應的資料類別中，並以 FIFO 的方式傳輸給下一級的功能。

responsePush 接收更新後的委託簿五檔價量後，將該資料包裝成數據流的資料型態並傳送給下一級的模組。

(三) pricingEngine

pricingEngine 模組的設計如下，

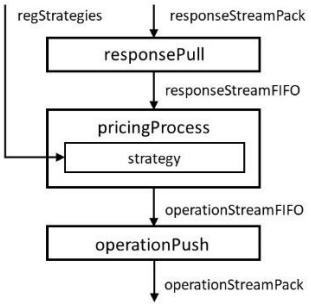

圖4 pricingEngine 模組架構

responsePull 接收包裝後的委託簿最新五檔價量資料後，將輸入資料拆封成對應的資料類別，並以 FIFO 的方式傳輸給下一級的功能。

pricingProcess 接收拆封後的最新委託簿資料後，將委託簿最新五檔價量輸入對應的交易策略中，作出增加、更新或取消委託等決策。決策的資料存取於對應的資料類別中，並以 FIFO 的方式傳輸給下一級的功能。

operationPush 接收策略決定的委託操作後，將該資料包裝成數據流的資料型態並傳送給下一級的模組 orderEntry。

四、實作結果

在模擬驗證上，feedHandler 模組以元大金融控股所錄製的交易所封包驗證其解碼功能，orderBook 模組以數筆委託簿變更條目資料驗證新增、刪除和修改的變更功能，pricingEngine 模組以數筆委託簿五檔價量驗證做出決策的功能。

在合成表現上，根據排程表計算出核心功能的延遲為134奈秒/每筆測試資料。以台灣期貨交易所每秒一萬至一萬五千個封包的傳輸頻率，延遲時間屬於可接受範圍，但是若考慮封包的前處理和送出委託仍需時額外的時間，仍希望延遲能夠進一步優化。暫存器合計使用32,299單位，查找表合計使用42,690單

位，根據 Alveo U50數據中心加速卡的規格(暫存器1,743k 單位、查找表(872k 單位)，兩者的使用率分別為1.85%和4.89%，屬於可接受範圍。

Modules & Loops	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
feedHandlerTop	-	-	-	-	-	dataflow	0	0	15866	28942	0
inputStreamHandler	0	0.0	-	0	-	no	0	0	2	31	0
MDPacketHandler	1	8.000	-	1	-	no	0	0	1001	1203	0
entry_proc	0	0.0	-	0	-	no	0	0	2	20	0
symbolLookup	-	-	-	-	-	no	0	0	2221	7730	0
eventHandler	0	0.0	-	1	-	yes(flp)	0	0	66	84	0

Modules & Loops	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
orderBookTop	10	100.000	-	2	-	dataflow	32	0	8803	7204	0
operationPull	1	10.000	-	1	-	yes(flp)	0	0	213	111	0
operationProcess	5	50.000	-	2	-	yes(flp)	32	0	1404	1467	0
entry_proc	0	0.0	-	0	-	no	0	0	2	20	0
responsePush	2	20.000	-	1	-	yes(flp)	0	0	2322	147	0

Modules & Loops	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
pricingEngineTop	11	110.000	-	2	-	dataflow	6	0	7630	6544	0
entry_proc	0	0.0	-	0	-	no	0	0	2	29	0
responsePull	1	10.000	-	1	-	yes(flp)	0	0	829	111	0
pricingProcess	6	60.000	-	2	-	yes(flp)	2	0	1568	1130	0
operationPush	2	20.000	-	1	-	yes(flp)	0	0	522	147	0
eventHandler	0	0.0	-	1	-	yes(flp)	0	0	66	84	0

圖5 模組合成報告

五、結論

(一) 分析與討論

根據排程表，可以發現資料流的讀寫具有最高的延遲，因此想降低模組整體的延遲，有兩個改善方向。第一個是減少讀寫的資料流長度，目前的設計是讀寫全部的委託簿變更資料，可以在不影響委託簿更新的情況捨去部分資料來增加讀寫速度。第二個是減少讀寫的次數，目前的設計是 orderBook 每一筆委託簿變更資料都會重新從記憶體讀取委託簿五檔價量，並向下一級寫出最新的委託簿五檔價量。可以修改成若同一個封包中包含複數筆委託簿變更資料時，因為欲變更之委託簿皆屬於同一商品，只須從記憶體對委託簿五檔價量做一次讀取，接著根據多筆委託簿變更資料多次修改變更後，再將最新的委託簿五檔價量做一次寫出，減少讀寫次數。

另外因為 AAT 屬於開放資源，為了增加普遍性和可閱讀性，功能設計分得比較精細，卻也增加許多資料傳輸的步驟和程序，進而影響到表現。因此可以嘗試合併部分模組或是功能，不僅減少資料流組裝和拆分的次數，降低延遲，也可以減少暫存器和查找表的使用量。

(二) 未來目標

主要為硬體驗證。由於上述模組皆為 AAT 的部份模組，因此欲對部分模組進行硬體驗證，需編寫主機的程式碼定義電腦端至硬體端的資料傳輸，這部分尚在學習當中，因此目前實作階段進行至軟體模擬和合成，硬體驗證則是後續主要目標。

其次是實作下一級模組 orderEntry。orderEntry 負責將策略決定的委託操作之資料組裝成 TCP 封包內容，格式需根據台灣期貨交易的規定，然而本人目前手邊沒有這方面的資料可以參考，因此先擱置這一模組的實作。若能取得相關資料，orderEntry 的實作也是後續目標之一。

參考文獻

[1] Taiwan Future Exchange, “逐筆行情資訊傳輸作業手冊 V1.0.0”, November 2019, pp.14-19.

[2] James F Kurose and Keith W Ross, “Computer Networking, A Top-Down Approach (6th edition)”, March 2012, pp.1-430 .

[3] Xilinx, “Accelerated Algorithmic Trading User Guide”, October 2021, pp. 29-41.

心得感想

透過這幾個月的專題實作，了解到完成一項題目的必經流程，從最初學習相關領域的知識、討論題目的方向，過程中安排每周的進度、嘗試新的想法並不斷遭遇和解決問題，最後統整與分析實作結果，每一個階段都是寶貴的經驗，使我受益良多。

最後，非常感謝過程中給予我幫助的所有人。馬席彬教授提供我很多專題方向上的建議和相關的參考資料，實驗室趙德昊學長也幫助我解決很多技術層面的問題，另外每周的進度會議上其他組別分享的資訊和建議也相當實用。再次感謝幫助我的所有人，使我能順利完成專題實作。