

東海大學

資訊工程研究所

碩士論文

指導教授：楊朝棟博士

使用高效能 Linpack 的虛擬化平台性能  
評估

Performance Evaluation of Virtualization Platforms Using  
High Performance Linpack

研究生：宋昱琳

中華民國一〇五年六月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 宋 昱 琳 所提之論文

使用高效能 Linpack 的虛擬化平台性能評估

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

召 集 人

人

張之山

簽章

委

員

黃國展

江輔政

許慶賢

指 導 教 授

楊朝棟

簽章

中華民國 105 年 6 月 27 日

# 摘要

隨著虛擬化技術應用興起，其中 VMware Workstation 與 vSphere / ESXi、KVM 為高可用性之虛擬化管理平台，因應各產業對於「提高管理效率、降低企業成本」的目標下，於單一伺服器中配置數台虛擬機群，更能夠有效的提高資源利用率，也可以滿足使用者對於不同作業系統的需求。本論文考量到伺服器在負擔建立虛擬機群所可能造成的儲存消耗量提高，資料重複性高，造成主機端的負載增加，進而影響到效能降低，故提出由不同架構，不同虛擬化管理平台的比較測試，使用 GUN 編譯器與 C++ 編譯器及運用 High Performance Linpack 與 Intel Linpack 進行測試，以取得最佳配置方案，能提供資訊管理人員於建置虛擬化環境前的參考依據，除了伺服器上的資源可以更充份的被利用外，也可以提高效能，更進一步去減少購置伺服器的數量，來降低企業成本，並且也可以節省能源，最後，於本文中所建議之 ESXi 環境下進行建立虛擬機群，可使虛擬機效能發揮到最佳化。

關鍵字：虛擬化、VMware、HPCC 效能計算、編譯器

# Abstract

With the rising of virtualization technology applications, VMware Workstation and vSphere / ESXi, KVM had used as high-availability virtualization management platform. In response to the target of various industries to "improve management efficiency, reduce costs", configuring the number of servers in a single virtual cluster, it's not only able to effectively improve the utilization of resources, but also to meet with those who make different operating systems. In this paper, for considering construct a virtual server in the cluster may cause the increasing of storage consumption, high data duplication, resulting in increased load on the host side, and cause performance degradation. Therefore, we proposed the comparison test of different architectures, different virtualization management platform. In order to obtain the best configuration, to provides the information of building virtualized environment. We've using of GUN compiler and C ++ compiler on different platform, and using High Performance Linpack and Intel Linpack for performance testing. It makes sufficient resources can be used more on the server, it can also improve efficiency, to further to reduce business costs by reducing the number of servers purchased, and can also save energy. For conclusion, using ESXi to create a virtual cluster can get the best performance.

Keyword : Virtualization Technology 、 VMware 、 HPC Challenge Benchmark Suite (HPCC) 、 Compiler



## 致謝詞

在東海的求學過程中，在此首要感謝指導老師楊朝棟教授，楊老師對於學生的品性與學業都有相當程度的要求，在嚴格中卻又不失對學生們的關懷，是一位很值得敬重與學習的老師，在課程方面都是最新最符合產業界所需的軟體應用，總能啟發學生多元思考、靈活運用，讓學生實際操作以了解其原理，並確切的將所學融合貫通加以變化，感謝楊教授二年來的教導與磨練；在艱辛的求學過程中，張廣欽學長是我另一位老師，每每遇到無法排除的問題，學長總是能運用各種比喻或是關連性來教導，在學長的身上有很多特質：謙遜、恆心與堅持，很值得學習。

並且感謝國立台北大學資訊工程學系張玉山教授、中華大學資訊工程學系許慶賢教授、國立台中教育大學資訊工程學系黃國展副教授、東海大學資訊工程學系江輔政副教授，諸位老師於口試過程中，給予寶貴的意見與指導，使本論文能更充份與完整。

能考上東海大學資訊工程研究所並順利取得學位，終於完成人生目標裡的其中一個願望，證明了「學海無崖，勤是岸」，只要對任何事都抱持著積極面對的態度，勇敢邁開步划向前，就有成功的機會。在這漫長的人生雖然難免會有風雨、或是遭遇挫折，但能從中學習到的經驗，都將成為茁壯自己的養份，相信只要肯付出、肯努力，沒有完成不了的。

最後，特別感謝家人的默默支持與體諒，以及男朋友的包容，因為常常太專注於課業，所以連跟他們聚餐的時間都沒有，總是匆匆忙忙的回個幾句話又繼續忙，心裡感到很抱歉，但，我很愛你們。

# Table of Contents

摘要	I
Abstract	II
致謝詞	III
Table of Contents	IV
List of Figures	V
List of Tables	VI
<b>1 簡介</b>	<b>1</b>
1.1 研究動機	1
1.2 論文目標與貢獻	2
1.3 論文架構	3
<b>2 研究背景與相關研究</b>	<b>4</b>
2.1 研究背景	4
2.1.1 虛擬化技術	4
2.1.1.1 主機型架構	6
2.1.1.2 裸機架構	7
2.1.2 虛擬化平台	8
2.1.2.1 VMware Workstation 12 工作站	8
2.1.2.2 VMware vSphere 6 / ESXi 6 虛擬管理程式	9
2.1.2.3 KVM 核心虛擬技術	9
2.1.3 高性能 Linpack 基準測試	10
2.1.4 編譯器	11
2.1.4.1 GNU 編譯器	13
2.1.4.2 Intel C++ 編譯器	14
2.1.5 基礎線性代數程式集	15
2.1.5.1 GotoBlas	15
2.1.5.2 OpenBlas	15
2.1.5.3 Intel MKL	15
2.1.6 信息傳遞介面標準	16
2.2 相關研究	16

<b>3</b>	<b>實驗環境與建置</b>	<b>18</b>
3.1	實驗架構	18
3.2	機器規格	23
3.2.1	主機板	24
3.2.2	中央處理器	24
3.2.3	硬碟	26
3.2.4	隨機存取記憶體	27
3.3	系統與軟體版本	28
3.3.1	系統版本	28
3.3.1.1	Windows	28
3.3.1.2	Ubuntu	30
3.3.2	虛擬化平台版本	31
3.3.2.1	VMware Workstation	32
3.3.2.2	VMware vSphere / ESXi	33
3.3.2.3	KVM	33
<b>4</b>	<b>實驗環境與結果</b>	<b>35</b>
4.1	測試架構	35
4.1.1	基礎測試環境架構	35
4.1.2	測試架構說明	35
4.2	理論值計算	37
4.3	實驗方法	38
4.3.1	HPL 安裝與實測	38
4.3.2	GotoBlas 與 OpenBlas 測試	38
4.3.2.1	GCC 編譯器執行測試	39
4.3.2.2	GCC 執行 Intel MKL 測試	39
4.3.2.3	ICC 編譯器安裝與測試	39
4.3.2.4	ICC 執行 Intel MKL 測試	39
4.3.2.5	Intel Linpack 測試	39
4.3.2.6	實驗加測組	40
4.3.2.7	數據彙整	40
4.4	實驗結果	40
4.4.1	HPL 實測結果	40
4.4.2	GCC 編譯器實測結果	42
4.4.3	ICC 編譯器實測結果	44
4.4.4	MKL 數學核心函式庫實測結果	46
4.4.5	Intel Linpack 實測結果	48
4.4.6	KVM 實測結果	49
4.4.7	分區分塊因子分析結果	50
4.4.8	各架構效能表現	51
4.4.8.1	A 架構	52
4.4.8.2	B 架構	53
4.4.8.3	C 架構	54
4.4.8.4	D 架構	55

4.4.8.5 E 架構 . . . . .	56
<b>5 結論與未來方向</b>	<b>57</b>
5.1 結論 . . . . .	57
5.2 未來方向 . . . . .	58
參考文獻	59
附錄	66
<b>A 安裝 Ubuntu</b>	<b>66</b>
A.1 Ubuntu 下安裝 KVM . . . . .	68
<b>B 安裝 VMware vSphere</b>	<b>74</b>
B.1 安裝 VMware vSphere . . . . .	74
B.2 於 VMware vSphere 6 下安裝 Ubuntu . . . . .	81
<b>C 安裝 VMware Workstation 12</b>	<b>90</b>
C.1 於 windows 7 安裝 VMware Workstation 12 . . . . .	90
<b>D 其他附屬安裝</b>	<b>93</b>
D.1 GCC 編譯器下載安裝 . . . . .	93
D.2 OPEN MPI 下載安裝與執行 . . . . .	95
<b>E 效能測試安裝與執行</b>	<b>99</b>
E.1 HPL 安裝與實測 . . . . .	99
E.2 GotoBlas 與 OpenBlas 測試 . . . . .	102
E.2.1 GCC 編譯器執行測試 . . . . .	102
E.2.2 GCC 執行 Intel MKL 測試 . . . . .	103
E.2.3 ICC 編譯器安裝與測試 . . . . .	103
E.2.4 ICC 執行 Intel MKL 測試 . . . . .	107
<b>F 測試數據統計表</b>	<b>108</b>
F.1 HPL 統計表 . . . . .	108
F.2 GCC 統計表 . . . . .	110
F.3 ICC 統計表 . . . . .	112
F.4 MKL 統計表 . . . . .	114
F.5 分區分塊因子統計表 . . . . .	116
<b>G Intel CPU Gflops</b>	<b>119</b>
G.1 Haswell、Sandy Bridge 指令集倍率對照表 . . . . .	119
G.2 E3-1230 V3 . . . . .	120
G.3 i7-3960X . . . . .	121

# List of Figures

2.1	主機型架構	6
2.2	裸機架構	7
2.3	編譯器的主要工作流程示意圖	12
2.4	執行檔工作流程圖	13
3.1	虛擬化架構分類	19
3.2	A 架構示意圖	20
3.3	B 架構示意圖	21
3.4	C 架構示意圖	21
3.5	D 架構示意圖	22
3.6	E 架構示意圖	23
3.7	中央處理器構成	25
3.8	Windows 畫面-E3	29
3.9	Windows 畫面-i7	29
3.10	Ubuntu 畫面-E3	30
3.11	Ubuntu 畫面-i7	30
3.12	VMware Workstation 介面	32
3.13	vSphere 介面	33
3.14	KVM 介面	34
4.1	實驗項目	36
4.2	各項變數定義	37
4.3	HPL 實測結果呈現圖-E3	41
4.4	HPL 實測結果呈現圖-i7	41
4.5	GCC 編譯器實測結果呈現圖-E3	42
4.6	GCC 編譯器實測結果呈現圖-i7	43
4.7	ICC 編譯器實測結果呈現圖-E3	44
4.8	ICC 編譯器實測結果呈現圖-i7	45
4.9	MKL 實測結果呈現圖-E3	46
4.10	MKL 實測結果呈現圖-i7	47
4.11	Intel Linpack 實測結果呈現圖-E3	48
4.12	Intel Linpack 實測結果呈現圖-i7	48
4.13	KVM 實測 HPL 結果-i7	49
4.14	KVM 實測 Intel Linpack 結果-i7	49
4.15	範例：A 架構分區分塊因子取出數	50
4.16	A 架構效能表現圖-E3	52

4.17 A 架構效能表現圖-i7 . . . . .	52
4.18 B 架構效能表現圖-E3 . . . . .	53
4.19 B 架構效能表現圖-i7 . . . . .	53
4.20 C 架構效能表現圖-E3 . . . . .	54
4.21 C 架構效能表現圖-i7 . . . . .	54
4.22 D 架構效能表現圖-E3 . . . . .	55
4.23 D 架構效能表現圖-i7 . . . . .	55
4.24 E 架構效能表現圖-i7 . . . . .	56
A.1 Ubuntu 安裝畫面 . . . . .	66
A.2 Ubuntu 登入畫面 . . . . .	67
A.3 終端機 Terminal . . . . .	68
A.4 查詢有無支援虛擬化技術 . . . . .	68
A.5 安裝 KVM . . . . .	69
A.6 確認 KVM 執行狀態 1 . . . . .	69
A.7 確認 KVM 執行狀態 2 . . . . .	69
A.8 KVM 管理程序 1 . . . . .	70
A.9 KVM 管理程序 2 . . . . .	70
A.10 建立虛擬機 1 . . . . .	70
A.11 建立虛擬機 2 . . . . .	71
A.12 建立虛擬機 3 . . . . .	71
A.13 建立虛擬機 4 . . . . .	72
A.14 建立虛擬機 5 . . . . .	73
B.1 ESXi 安裝畫面 . . . . .	74
B.2 ESXi 身份驗證畫面 . . . . .	75
B.3 ESXi 載入設定畫面 . . . . .	76
B.4 ESXi 下載網址畫面 . . . . .	77
B.5 vSphere Client 安裝畫面 . . . . .	78
B.6 vSphere Client 桌面 icon . . . . .	79
B.7 vSphere Client 驗證授權碼畫面 1 . . . . .	79
B.8 vSphere Client 驗證授權碼畫面 2 . . . . .	80
B.9 vSphere Client 驗證授權碼畫面 3 . . . . .	80
B.10 結束 ESXi 畫面 . . . . .	81
B.11 新增：clock.stdtime.gov.tw 1 . . . . .	81
B.12 新增：clock.stdtime.gov.tw 2 . . . . .	82
B.13 新增：clock.stdtime.gov.tw 3 . . . . .	82
B.14 新增：上傳 iso 檔案 1 . . . . .	83
B.15 新增：上傳 iso 檔案 2 . . . . .	83
B.16 建立新的虛擬機器 1 . . . . .	84
B.17 建立新的虛擬機器 2 . . . . .	84
B.18 虛擬機器組態 . . . . .	85
B.19 虛擬機器的名稱和位置 . . . . .	85
B.20 虛擬機器目的地儲存區 . . . . .	86
B.21 編輯虛擬機器設定 . . . . .	87

B.22	設定安裝來源檔	87
B.23	切換至主控台	88
B.24	設定 BIOS	89
B.25	Ubuntu 安裝畫面	89
C.1	VMware Workstation 安裝畫面	90
C.2	VMware Workstation 完成安裝畫面	92
C.3	VMware Workstation 啟動畫面	92
D.1	安裝建置畫面	94
D.2	查看安裝狀況畫面	94
D.3	OPEN MPI 下載安裝畫面	95
D.4	查看目錄內容	95
D.5	安裝 openMPI	96
D.6	環境變數	97
D.7	配置生效	97
D.8	執行測試 (啟用範例)	98
E.1	下載與安裝 GCC	99
E.2	檢查安裝狀態	100
E.3	HPL scripts 批次檔下載	100
E.4	GCC Compiler 編譯函式庫過程	101
E.5	取得效能基準	101
E.6	執行 GCC 編譯 GotoBlas 畫面	102
E.7	執行 GCC 編譯 OpenBlas 畫面	102
E.8	執行 GCC 編譯 Intel MKL 畫面	103
E.9	Intel 官網申請畫面	104
E.10	Intel 官網資料登錄畫面	104
E.11	執行 ICC 編譯 GotoBlas 畫面	105
E.12	執行 ICC 編譯 OpenBlas 畫面	106
E.13	執行 ICC 編譯 Intel MKL 畫面	107
F.1	HPL 數據統計-E3	108
F.2	HPL 數據統計-i7	109
F.3	GCC 數據統計-E3	110
F.4	GCC 數據統計-i7	111
F.5	ICC 數據統計-E3	112
F.6	ICC 數據統計-i7	113
F.7	MKL 數據統計-E3	114
F.8	MKL 數據統計-i7	115
F.9	分區分塊因子數據統計-E3	116
F.10	分區分塊因子數據統計-i7	117
F.11	分區分塊因子數據統計-i7-KVM	118
G.1	Haswell、Sandy Bridge 指令集倍率對照表	119
G.2	Intel CPU E3-1230 V3 Gflops	120

G.3 E3-1230 V3 Haswell 確認 . . . . . 121  
G.4 Intel CPU i7-3960X Gflops . . . . . 121





# List of Tables

2.1	虛擬化具體差異 . . . . .	5
2.2	VMware Workstation 單一虛擬機資源配置最高規格 . . . . .	8
2.3	VMware vSphere ESXi 單一虛擬機資源配置最高規格 . . . . .	9
2.4	測量每秒浮點運算次數單位對照表 . . . . .	11
2.5	GNU 開發工具常用工具清單 . . . . .	14
2.6	Intel Parallel Studio XE 2016 之功能 . . . . .	14
3.1	E3 主機測試架構 . . . . .	18
3.2	i7 主機測試架構 . . . . .	19
3.3	KVM 測試組架構 . . . . .	19
3.4	E3 硬體規格 . . . . .	23
3.5	i7 硬體規格 . . . . .	24
3.6	Intel E3 處理器規格 . . . . .	26
3.7	Intel i7 處理器規格 . . . . .	26
3.8	網格運算記憶體需求計算公式 . . . . .	27
3.9	網格運算記憶體需求試算 . . . . .	27
3.10	實驗架構記憶體需求總量計算公式 . . . . .	27
3.11	實驗所需的記憶體需求預估 . . . . .	27
4.1	理論值計算公式 . . . . .	37
4.2	效能比變數名稱定義 . . . . .	37
4.3	E3 CPU 理論值計算 . . . . .	38
4.4	i7 CPU 理論值計算 . . . . .	38
4.5	效能與理論值計算比例公式 . . . . .	38
4.6	E3 分區分塊因子落點 . . . . .	50
4.7	i7 分區分塊因子落點 . . . . .	51
4.8	i7-KVM 分區分塊因子落點 . . . . .	51

# Chapter 1

## 簡介

### 1.1 研究動機

VMware Workstation 是基於宿主作業系統的虛擬化，十分依賴宿主操作系統的穩定性，因一般實體機於安裝時多數已具備各種所需驅動程式，故建立虛擬機時會受到實體設備規格所限制；而 ESXi 是基於裸機虛擬化技術，經由虛擬平台模擬出所需之設備規格，但需了解該物理機設備的條件限制，以確保其可行性，唯安裝驅動程式時需留意該實體機的相容性，以免產生不相容或是可能導致系統及應用程序出現無法運作、執行等情況，當一伺服器之效能可充份發揮時，意謂著資源是充足的，可再評估增加虛擬機數量，進而減少配置伺服器之數量，來降低成本，由管理層面來看，也更具系統性。

虛擬機群的配置與節點的分佈則視伺服器端可提供的資源而定，故其儲存消耗皆由所對應之伺服器負擔，在資源的取得與擴充部份，除了要在雲端配置新節點或是尋求共享資源來降低配置虛擬機群所產生的儲存消耗量與管理成本…等問題外。另外尚需考量到幾種可能影響效能之因素，例如配置不當而造成虛擬機群之資源不均導致無法充份發揮其效能、軟/硬體之相容性…等因素，本研究將使用不同架構（Hosted hypervisor、Bare-Metal hypervisor）與不同虛擬化平台作測試，求出當中效能最佳者，以作為系統管理人員在伺服器上配置虛擬機群時的參考依據。

在先期試驗當中，我們得知當逐一建立虛擬機群，同時也會影響主機端的儲存消耗量，其問題在於儲存的資料重複性高，因而會造成主機端的負載增加。本論文使用主機型架構及裸機架構，並使用不同虛擬化平台進行效能測試，以取得其效能差異。

## 1.2 論文目標與貢獻

本論文目標在於測試在各不同虛擬化管理平台之中，且在相同的 Ubuntu 作業系統中進行效能的差異比較與評估，使用 GUN 編譯器與 C++ 編譯器二種編譯器，運用 High Performance Linpack 與 Intel Linpack 進行測試，來獲得更準確的效能配置指標，並提出可供系統管理人員就的現有設備資源，作最佳配置之參考依據。當資源有效的被利用與易於分配管理時，效能評估亦為重要的考量，如運用本論文中所提出的最佳化配置，能夠使虛擬機更充份的發揮效能，進而減少設備的配置成本與有效的提高資源利用率。此外，系統管理人員於建置虛擬化架構時，一併將總體效率分配及個體效能使用率的比重進行評估，不僅可確保所提供的服務品質，也可以滿足眾多使用者對不同作業系統及規格、效能的需求。整體來看，可減少配置設備、提高設備效能與資源利用率、更靈活的調度與共享資源，形成三方（企業、系統管理人員、使用者）皆滿足的目標。

本論文中 Host-based Architecture 運用 VMware Workstation、vSphere、KVM；Bare Metal Architecture 運用 VMware ESXi；另於實體機上直接安裝 Ubuntu 作業系統測試，將所取得效能數據作為基準，藉此來比較不同架構及虛擬化平台的效能差異，藉以作為資訊人員於配置時納入最優參考依據。

### 1.3 論文架構

本論文主要在介紹虛擬機配置架構對效能之影響評估，並運用不同架構測試效能，其測試結果可作為系統管理人員建立、配置虛擬機群時的前期參考依據。第一章簡述相關研究過程與結果，再於虛擬化的現況中尋求可改善，提昇的方面著手研究，並且從中延伸出本論文的研究目標及方向；第二章介紹相關研究背景及其各項相關虛擬化技術概念、本文中將使用之運用工具進行說明簡述；第三章節中將描述論文中使用的架構概觀與測試機器的資源配置規格；第四章實驗環境配置、運用之測試工具與實驗測試數據產出與比較其差異點；最後於，第五章將對於本論文之實驗過程、結果，經彙整後作出最終結論，並且從中延伸出未來的研究方向。

# Chapter 2

## 研究背景與相關研究

### 2.1 研究背景

#### 2.1.1 虛擬化技術

虛擬化打破實體限制的疆界，可以於單一伺服器虛擬出多個系統，充份提高設備利用率，亦可整合多台伺服器的資源作靈活的運用管理、移轉等，使運算資源變得更為彈性，不再將資源受限於單一伺服器，系統管理人員透過監控器可有效將伺服器資源作統合分配，使資源平均分配於各虛擬機，提高資源利用率。未經雲端共享的情況下，虛擬出的機器仍是以實體機的 CPU 運算能力為上限，實際上亦是使用實體機所具備的資源；經由雲端作伺服器的共享連結，依使用者需求可在不同伺服器中切換可用之虛擬機，故可減輕單一伺服器的負載，此外，虛擬機可各自擁有一 IP 位址，使用者可經由任一實體機連結使用，系統管理人員亦經由網路來處理使用者面臨之問題並提出解決方案，在遠端線上處理其問題，能降低時間成本，便於在伺服器上的資源作統籌分配與調度。

對於企業端部份，虛擬化技術滿足了企業期望改善系統與資源管理效率的目標，另一方面又能不增加營運支出；對於系統管理人員而言是更可以有效地整合分散的資源、方便統一管理的需求、提高資源使用率與資源運用的靈活度；虛擬化的優點：有效整合與分配資源、降低管理層面的複雜度、節能省電符合

綠能政策，更重要的是能夠為企業降低營運管理成本，省下不少維護成本與伺服器費用。

在虛擬化的分區上，伺服器主機端可視其資源配置虛擬機群，或因應不同版本的作業系統需求作安裝配置，並可以隨需求調動虛擬機資源，滿足使用者需求，能夠為企業省下採購軟、硬體設備費用；彈性與靈活的運用資源及大幅提升運作效率的特性，系統管理人員藉由主控台可以監控所有實體機與虛擬機的運作狀況、資源利用率…等，作統籌管理分配；虛擬化的配置能有效減少伺服器的數量，簡化管理架構，增加資源利用率。

主機型架構與裸機型架構皆是藉由使用虛擬機器監控器去建立一台或數台虛擬機，當中的共同性為皆佔用實體機的儲存資源，主機型架構因硬體設備從實體機進行導入與利用，可運用之資源受到限制，但穩定性較佳；相形之下，裸機型架構可依對硬體設備的需求去作修改與任意變動，但對於硬體設備所需之軟體部份，需要特別留意其相容性問題。[1]

TABLE 2.1: 虛擬化具體差異

項目	主機型架構	裸機架構
說明	Hosted hypervisor	Bare-Metal hypervisor
資源	實體機	實體機
安裝位置	作業系統上	裸機
硬體資源	實體資源	實體資源與虛擬資源
軟體需求	設備商提供	視實體機需求增減
軟體相容性	佳	需自行確認相容性問題

本文中所使用的 ESXi 或 vSphere 虛擬化平台用詞，皆是指相同的虛擬管理程序，vSphere 應用於主機型架構，ESXi 則應用於裸機型架構。

### 2.1.1.1 主機型架構

主機型架構 (Hosted hypervisor)，在虛擬機的建立時，模擬出基於實體機上的資源上限，虛擬機器監控器可以將實體硬體及所需驅動程式作導入的動作，例如記憶體、CPU、CD-ROM...等，而且也不需要特殊額外安裝啟用軟體，虛擬機所需的硬體設備來源取得，基本上皆由實體機實際上的設備所提供，對於建立虛擬機是更快速便捷的。

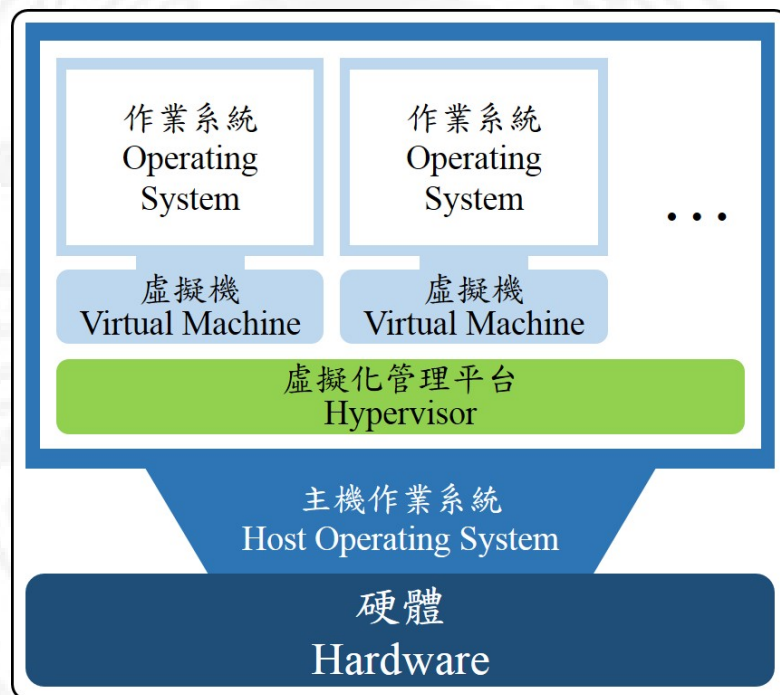


FIGURE 2.1: 主機型架構

### 2.1.1.2 裸機架構

裸機架構 (Bare-Metal hypervisor)，直接經由裸機安裝 Hypervisor，由虛擬機器監控器監控底層硬體的存取，能夠自由建立所需要的各類客體作業系統，視使用需求狀況，設定各種硬體設備及週邊裝置，滿足使用者不同的需求，雖然可利用之硬體設備的層面更廣，但是仍需留意所需設備規格是否有相容性問題。

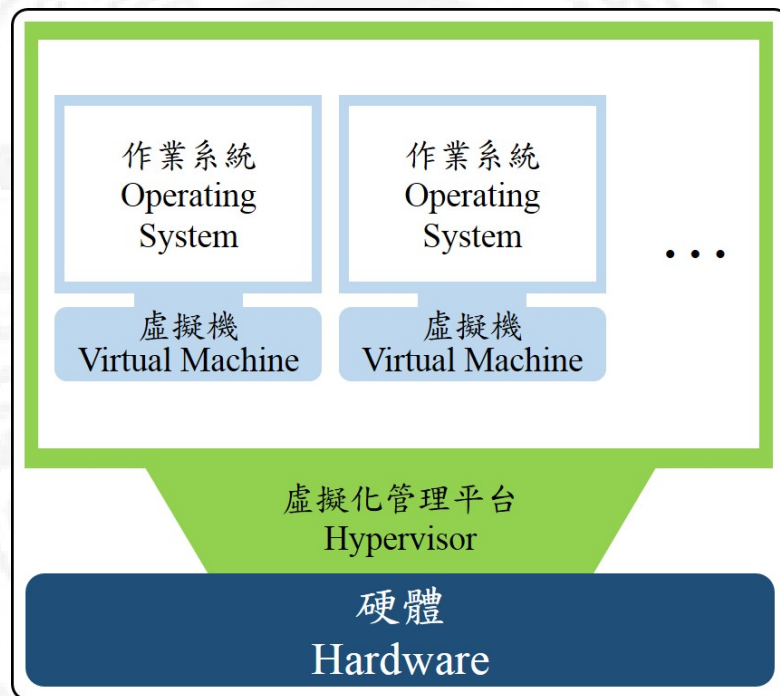


FIGURE 2.2: 裸機架構



## 2.1.2 虛擬化平台

### 2.1.2.1 VMware Workstation 12 工作站

VMware 虛擬化方案，可將作業系統和應用軟體與底層硬體分離，以共用的儲存池概念把多台伺服器與網路聚合後，按照需求分配資源給應用程式，提高資源利用率；VMware Workstation 支援 Windows 和 Linux、處理器和硬體，可連接 VMware vSphere 和 vCloud Air，對於虛擬機交叉相容性極佳，並可依據需求建立完善的虛擬化管理程序來執行 Hyper-V 與 VMware Sphere；單一虛擬機的資源配置上最多可達 16 個虛擬處理器（Virtual Central Processing Unit，VCPU），以及 8 TB 虛擬硬碟（Virtual Machine Disk，VMDK）和 64 GB 虛擬記憶體（Virtual Random Access Memory，VRAM）。

TABLE 2.2: VMware Workstation 單一虛擬機資源配置最高規格

硬體	規格上限
虛擬處理器 (Virtual Central Processing Unit, VCPU)	16
虛擬硬碟 (Virtual Machine Disk, VMDK)	8 TB
虛擬記憶體 (Virtual Random Access Memory, VRAM)	64 GB

快照 (Snapshot) 功能可將虛擬機狀態作還原點的建立，如虛擬機被病毒入侵時，可使用此功能將虛擬機還原至指定狀態下；如需建立數個相同規格之虛擬機時，於各項初始設定及所需軟體安裝完成後，僅需使用快照 (Snapshot) 功能中的複製 (Clone) 功能進行快速建立，對於啟動的虛擬機越多，佔用的物理機資源也愈多的情況下，其使用父系連結建立虛擬機也是節省儲存資源的一大特色；在透過該平台可於不影響主機的情況下，自訂虛擬網絡組態，輕易地經由網絡串聯起各虛擬機成為群組，隨時隨地都能從網絡中遠端取得連接運行中的虛擬機來使用，更可以充分利用本地端電腦硬體及善用在雲端上的共享資源。[2]

### 2.1.2.2 VMware vSphere 6 / ESXi 6 虛擬管理程式

VMware vSphere Hypervisor (ESXi) 虛擬管理程式，其 ESXi 是直接安裝在實體伺服器上，能分割成多個邏輯伺服器（亦稱為虛擬機），同一實體機中的每個虛擬機可同時共用相同的實體資源，也節省下不少配置硬體設備的費用，並具有即時移轉能力，可在不停機的狀態下將作用中的虛擬機在實體伺服器之間進行移轉，且隨過簡化的部署與組態設定，也降低了管理的成本；單一虛擬機的資源配置上最多可達 128 個虛擬處理器 (Virtual Central Processing Unit, VCPU)，以及 62 TB 虛擬硬碟 (Virtual Machine Disk, VMDK) 和 4 TB 虛擬記憶體 (Virtual Random Access Memory, VRAM)。

TABLE 2.3: VMware vSphere ESXi 單一虛擬機資源配置最高規格

硬體	規格上限
虛擬處理器 (Virtual Central Processing Unit, VCPU)	128
虛擬硬碟 (Virtual Machine Disk, VMDK)	62 TB
虛擬記憶體 (Virtual Random Access Memory, VRAM)	4 TB

所有的 vSphere 主機映像統一儲存於 Auto Deploy 資料庫，管理員可依使用者所定義的規格自動建立新的主機，達到快速便捷的要求，管理員可指定 IP 範圍供使用者連線至主機來取得服務。[3] [4] [5] [6] [7]

### 2.1.2.3 KVM 核心虛擬技術

Kernel Virtual Machine (KVM)，為開源 Linux 虛擬管理平台之一，適用於主機型架構，需要實體機 CPU 支援虛擬化，其優勢為啟動虛擬機之速度較 vSphere / ESXi 來的迅速，但主要缺點有：因為發行於 Linux 版本中，如需於 Windows 作業系統下使用，則需額外安裝虛擬平台再安裝 Linux 作業系統，方可使用，且操作界面的親和度稍差，故管理者的技術門檻相對提高，但因其為開源軟體，可降低企業虛擬化部署成本。

KVM 建立虛擬機之上限取決於實體機的資源，所以實體機之資源分配將影響 KVM 中所有虛擬機的效能，管理人員可經由 KVM 介面中所提供之管理工具進行資源調度。

### 2.1.3 高性能 Linpack 基準測試

HPCC (HPC Challenge Benchmark) 為美國國防部先進研究計畫署之高效能技術電腦計畫中所提出，主要是用於評估效能，其七大測試項目：

一、HPL：利用與 Linpack-HPC 相同線性方程組，測量評估運算性能。

二、PTRANS：矩陣轉換並行程式，衡量系統訊息傳遞效能。

三、STREAM：量測內存性能。

四、Random Access：量測內存隨機更新速率。

五、Latency and Bandwidth：測量網絡性能，並測量節點間的傳輸性能。

六、DGEMM：量測矩陣相乘的運算效能。

七、FFTE：對 FFT (高速傅立葉變換) 運算性能、內存總體性能測量。

Linpack 起於 1974 年 4 月，美國 Argonne 國家實驗室應用數學所主任 Jim Pool 所提出的計畫案，原始目的在於建立一套專門解線性系統問題之數學軟體，經送到國家科學基金會 (National Science Foundation) 審核通過後，由美國政府出資，邀集高效能計算領域的專家組成團隊，並推派田納西大學的 Dr. Jack Dongarra 擔任計畫主持人，最終目的為發展出國際通用的標竿基準。

論文當中所使用的 HPL Linpack 基準測試 (High-Performance Linpack Benchmark, HPL) 則為其中之一，並行運算 (Parallel computing) 意謂著將一工作經分割後交給數個處理器同時執行，以獲得較快的結果。主要目的為測量系統的浮點運算能力，以最佳效能之矩陣大小為基礎，當中包含網格運算 (Grid computing) 與計算峰值，計算峰值的指標又分為理論浮點峰值和實測浮點峰值，指的是計算機每秒峰值速度 (每秒浮點運算次數) (Floating-point operations per second, FLOPS)，峰值速度主要以 CPU 的主頻所決定。目前大部分的處理器中都有浮點運算器，故每秒浮點運算次數所量測到的值，實際上就是浮點運算器的執行速度。[8] [9] [10] [11] [12] [13]

TABLE 2.4: 測量每秒浮點運算次數單位對照表

單位	說明
Mflop/s (MegaFLOPS)	每秒進行 $10^6 = 100$ 萬次浮點運算。
Gflop/s (GigaFLOPS)	每秒進行 $10^9 = 10$ 億次浮點運算。
Tflop/s (TeraFLOPS)	每秒進行 $10^{12} = 1$ 萬億次浮點運算。
Pflop/s (PetaFLOPS)	每秒進行 $10^{15} = 1$ 千萬億次浮點運算。
Eflop/s (ExascaleFLOPS)	每秒進行 $10^{18} = 100$ 億億次浮點運算。
GB/s (Giga)	每秒傳送 $10^9$ 位元組資料。

### 2.1.4 編譯器

1952 年葛麗絲·霍普在 UNIVAC I 上開發出第一套編譯器 A-0 系統，能將程式原始碼編譯為機器碼。1957 年由 IBM 的約翰·巴科斯所領導的 FORTRAN 則是第一套最完整具備編譯功能之編譯器。

編譯器 (Compiler)，為一電腦程式，其主要目的是將人們所使用的某種程式語言攢寫之程式碼 (原始碼 source code)，經由編譯器轉換成為可供電腦能夠解讀與執行的目標語言 (低階機器碼 machine code)，其目標語言又稱為執行檔。[14] [15]

編譯器主要的工作流程：

一、原始碼 (Source code)：

一般為高階語言 (High-level language)，如 C、C++、Java 等。

二、預處理器 (Pre-process)：

移除程式內的註解與框架…等，並確認語法，並可一次找出程式中不合文法的部份，如果沒有符合的話，編譯器會出現警告。

三、編譯器 (Compiler) 和組譯程式 (Assembler)：

將輸出的組合語言程式檔，轉換為處理器可以識別的二進制機械碼，產生目標檔 (object file)。故組合語言轉成機器碼的工具又稱為組譯器 (assembler)。

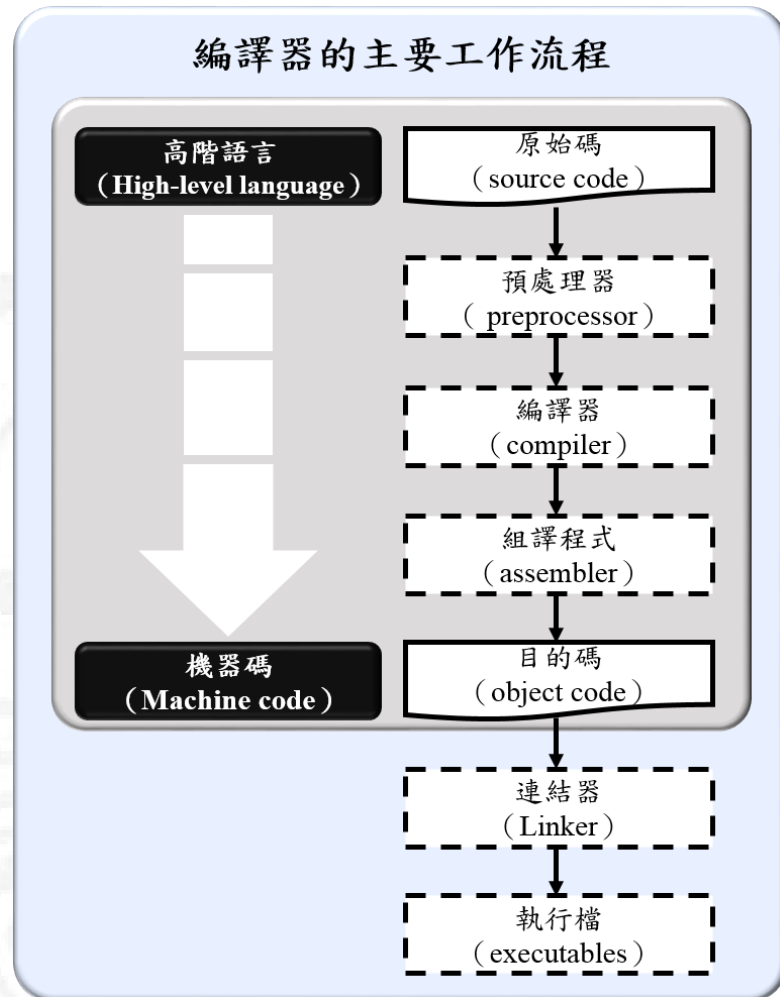


FIGURE 2.3: 編譯器的主要工作流程示意圖

#### 四、目的碼 (Object code)：

指編譯器或組譯器處理原始碼後所生成的代碼，由機器碼或接近於機器語言的代碼所組成。組織後成為目的檔 (object file) 為存放目的碼的檔案，亦稱作二進位檔案 (binaries)。且目的檔通常會包含符號表 (symbol table)、重新定位定址表 (relocation table) …等資訊，以供連結器 (linker) 及共享函式庫 (shared library) 執行時的資訊進行連結。

#### 五、連結器 (Linker)：

主要的工作是將多個由編譯器生成的目標文件與函式庫 (library) 進行連結，當指令與資料區塊重新組織排列後，成為一個或多個連結產生單一執行檔，或是整合多個目標檔成為一函式庫以供其他程式使用；所謂函式庫，即亦將繁複性

較高之程式，匯整集合為一，可被呼叫來執行某一特定功能函數，提供程式設計者方便使用。

#### 六、執行檔 (Executables)：

當編譯完成後，執行檔將存於硬碟上，執行檔內除了有程式的資訊外，尚有設定值資料、偵錯資料…等等。

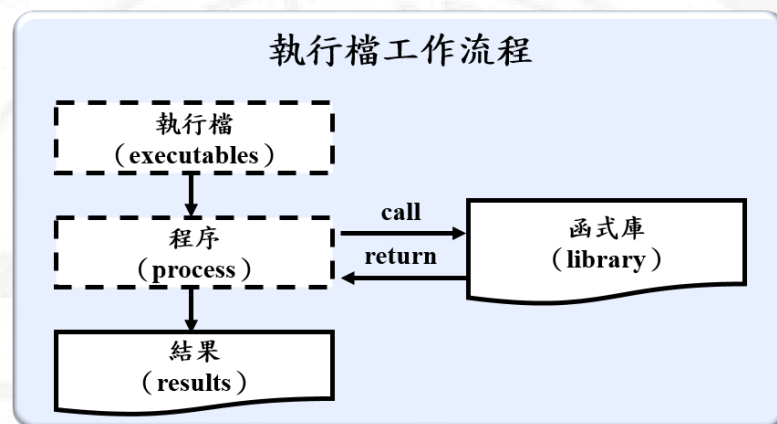


FIGURE 2.4: 執行檔工作流程圖

#### 2.1.4.1 GNU 編譯器

GNU 編譯器 (GNU Compiler Collection, GCC) 為 GNU 開發之編程語言編譯器，以 GPL 及 LGPL 授權條款所發行的自由軟體，1985 年由理察·馬修·斯托曼開始發展，現由自由軟體基金會負責維護工作。為處理 C 語言、C++、Fortran、Pascal、Object-C、Java, 以及 Ada…等其他語言；GNU 開發工具是由 GNU 計劃 (GNU Project) 中的數個程式開發工具所集合而成，能完成編譯、組譯和連結等步驟，還有具備有自動化設定、自動化編譯、除錯工具等功能。

GCC 被認為是跨平台編譯器的標準，也是最常被直接使用的，有別於一般侷限於特定系統與執行環境的編譯器，GCC 在所有平台上都使用同一個前端處理程式，產生一樣的中介碼，該中介碼在各個其他平台上使用 GCC 編譯，有很大的機會可得到正確無誤的輸出程式。[16] [17] [18]

TABLE 2.5: GNU 開發工具常用工具清單

英文名稱	中文名稱
GNU make	自動化編譯工具
GNU Compiler Collection (GCC)	編譯器套件
GNU Binutils	二進制工具
GNU Debugger (GDB)	除錯器
GNU Build System	自動化程式建構系統

#### 2.1.4.2 Intel C++ 編譯器

Intel C++ 編譯器 (Intel C++ Compiler, ICC), Intel Parallel Studio XE (Parallel Studio XE 2016- Cluster Edition for Linux) 是一套高性能並行運算編譯工具, 包含 Intel C/C++ 和 Fortran 編譯器、採用向量化和線程技術快速編碼 (記憶體和線程調優測試器, 可識別記憶體漏洞和內存分配錯誤), 也是集成 Intel 集群和 HPC 分析工具的套件, 兼容於 Windows\*、Linux\* 和 OS X\* 作業系統, 目的在於滿足性能、容量和效率的需求。

Intel Parallel Studio XE Cluster, 以 Intel 架構為基礎的叢集運算軟體, 所使用並行計算的應用程序介面 (Message Passing Interface, MPI) 可多台 PC 連結運算 (圖形/訊號處理、數據壓縮、加密、字串與符號處理), 並支援多核心運算。

由英特爾公司研發, 針對 Intel 處理器進行優化所設計出之編譯器, 但是對於非 Intel 但是相容於 Intel 架構的處理器, 如 AMD 處理器, 這個編譯器就無法產出最佳化的代碼。可編譯 C 語言與 C++, 另外也提供了多執行緒支援 (自動平行、OpenMP), 讓程式開發者可以用執行緒 (Thread) 的概念來開發平行計算程式。Table 2.6 僅列出本論文實驗所需相關 Intel Parallel Studio XE 2016 之功能。[19] [20] [21] [22] [23] [24] [25] [26]

TABLE 2.6: Intel Parallel Studio XE 2016 之功能

英文名稱	中文名稱
Intel C++ Compiler	英特爾 C++ 編譯器
Intel Math Kernel Library	英特爾數學核心函數庫 (MKL)
Intel MPI Library	英特爾 MPI 庫
Intel LINPACK Benchmark	英特爾 LINPACK 基準測試

### 2.1.5 基礎線性代數程式集

基礎線性代數程式集 (Basic Linear Algebra Subprograms, BLAS) 最初於 1979 年發布，其子程序中提供標準的建構模塊，主要是用來執行向量運算與矩陣乘法 (GEMM)，於程序中必須與 C 語言與 Fortran 編譯器配合使用來實現性能優化，被視為是應用程式介面 (API) 標準，並使用於建立更大的 Linpack 基準測試程序庫，對 Linpack 運算的成績有相當的助益。下面小節將介紹本論文中使用到主要的 BLAS。

#### 2.1.5.1 GotoBlas

高性能多核心 BLAS 庫，德克薩斯高階計算中心 Kazushige Goto 所開發，但已停止了活躍開發，後繼者為 OpenBlas。

#### 2.1.5.2 OpenBlas

高性能多核心 BLAS 庫，繼任 GotoBlas 的開源 BLAS 的實作，主要由中國科學院軟件研究所並列軟體與計算科學實驗室進行開發。

#### 2.1.5.3 Intel MKL

英特爾數學核心函數庫 (Intel Math Kernel Library)，為經過高度優化的 BLAS，專門使用在對於性能要求高的數學工程、科學、金融相關等領域，提供高度線程化的數學常式，此函式庫中包含了有 LAPACK、線性代數 (稀疏矩陣解算器)、快速傅立葉變換 (FFT)、向量數學庫 (VML) 等，跨平台 Windows / Linux 相容，支援 Pentium, Intel Core 與 Itanium 系列。



### 2.1.6 信息傳遞介面標準

信息傳遞介面標準 (Message Passing Interface, MPI) 是基於 MPI 論壇，該參與組織超過 40 個，組成人員包括大學研究員、政府人員、軟體開發人員與供應商，目標是 MPI 高性能，作為提高可擴展性和可移植性、建立跨語言信息傳遞程序之標準，用於電腦叢集與超級電腦平行計算的應用程序介面 (Application Programming Interface, API) 與並行計算中，本標準定義核心函式庫的語法和語義，提供簡單、靈活、高移植性的介面，使用者可以於多個不同的系統上進行高性能信息傳遞。[27] [28] [29] [30]

## 2.2 相關研究

根據相關虛擬化研究指出 IT 基礎設施之虛擬化可以整合與匯集資料，在不同的應用程序間進行資源的共享與分配，於一個實體主機中之虛擬化管理平台上，可提供具有支援多個作業系統的環境，藉由資源的使用率提昇，減少伺服器數量，提高了管理效率與成本效益、節能、易於移轉資源的各種優點。綜合上述優點，本論文以充份運用伺服器資源為目標進行不同虛擬化平台與交叉式平台架構測試。

本文參考 [31] Abdellatif Elsayed 對於虛擬機管理程序之穩定性與在密集的建立、使用虛擬機下對效能影響程度的介紹。[32] E. Angerson 等人利用 LAPACK 來設計與實現一個最佳高性能的線性代數庫，基於衡量 LINPACK 的最佳表現，於調整矩陣規模中可窺見一二，此外，亦參考來自 [33] Sebastien Varrette 等人之對於不同虛擬機管理程序的 HPL 測試，其中，除了比較不同虛擬化管理程序的效能差異，本論文以無虛擬化之環境實測作為參考基準。並使用相同之 ESXi 虛擬化平台，增加不同虛擬化平台進行實測與使用 Intel Linpack 進行複測，以驗證在不同虛擬化平台是否影響效能及提供詳細數據以便於達到符合實際需求之參考 [34] [35]。[36] Ou ZhiQiu 與 Zhou Yue 於 Research on application of virtualization in network technology course 一文中所提到 VMware Workstation 於學術單位之應用是以可減少系統環境的複雜設置及

設備費用。於大規模建立虛擬機群中，其優勢有快速、效率、方便遷移、易於管理等 [37] [38] [39] [40]。本論文中以不同虛擬架構、不同虛擬化平台進行比較後，所提出之最優虛擬化配置，可作為管理人員於配置虛擬化環境時，也考量到伺服器之規格與架構之優化。

關於虛擬化的平台種類繁多，本論文中除了運用 VMware Workstation 以及 vSphere / ESXi 之外，再於二主機中選出效能表現最佳者，另行安裝 KVM 來進行比較，開源的 KVM 用於 Linux 作業系統，可節省成本開銷，於管理層面來看，其介面的功能較少，所以使用上的便利性及維護成本，也需要納入虛擬機配置的考量之一。[41] [42] [43] [44] [45]

# Chapter 3

## 實驗環境與建置

### 3.1 實驗架構

在 E3 與 i7 主機上安裝實驗的四個架構，分別使用配置於三顆硬碟中，硬碟編號 I 配置架構 A、B；硬碟編號 II 配置架構 C；硬碟編號 III 配置架構 D；於實驗完成後取得效能表現最佳之主機進行加測 KVM。於部署時，需留意配置規格，以符合最終測試機需求規格，基礎規格相對應之要求（記憶體 24G；硬碟容量 100GB），於 Deploy 欄中表明安裝階層，且測試機位於 Deploy 層之最下層作業系統下，其部署之預設架構如下。

TABLE 3.1: E3 主機測試架構

CPU : Intel Xeon E3-1230 V3 3.30GHz(64 位元)

Test No.	Deploy	Finally	硬碟編號
A	Windows \ VMware Workstation \ vSphere	Ubuntu	I
B	Windows \ VMware Workstation	Ubuntu	I
C	ESXi	Ubuntu	II
D	Ubuntu	Ubuntu	III

TABLE 3.2: i7 主機測試架構

CPU : Intel Core i7-3960X 3.30GHz(64 位元)

Test No.	Deploy	Finally	硬碟編號
A	Windows \ VMware Workstation \ vSphere	Ubuntu	I
B	Windows \ VMware Workstation	Ubuntu	I
C	ESXi	Ubuntu	II
D	Ubuntu	Ubuntu	III

TABLE 3.3: KVM 測試組架構

CPU : Intel Core i7-3960X 3.30GHz(64 位元)

Test No.	Deploy	Finally	硬碟編號
E	Ubuntu \ KVM	Ubuntu	III

依虛擬化的架構來區分，架構 A、B 為主機型架構，架構 C 為裸機架構，實驗對照組 D 則無虛擬化，架構 E 為實驗加測組。

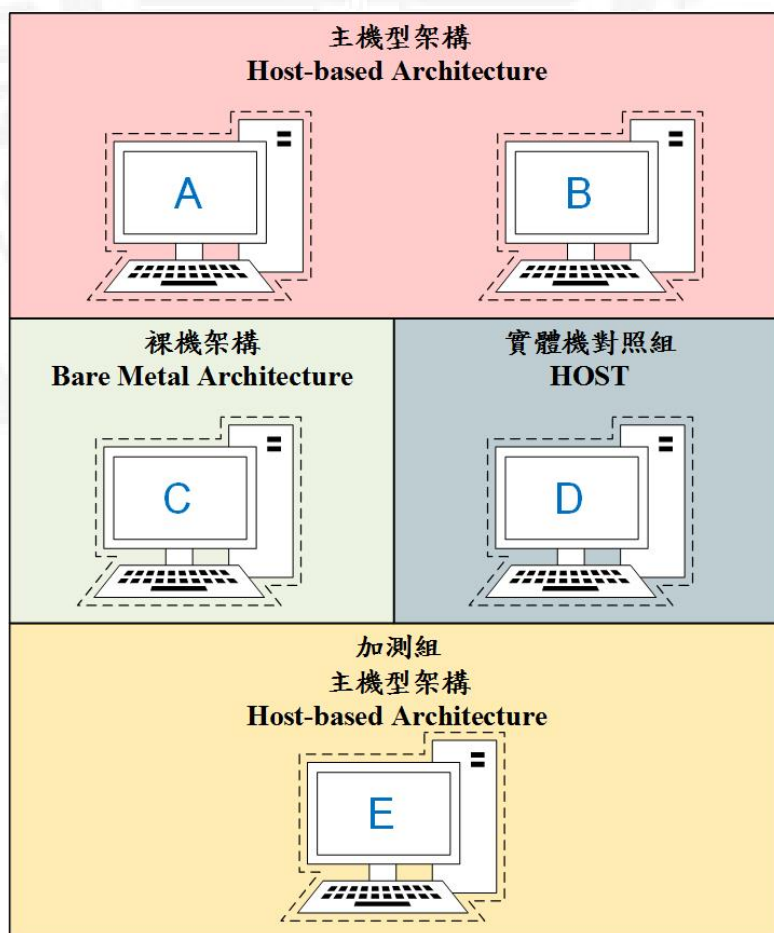


FIGURE 3.1: 虛擬化架構分類

A 架構設置：於 Windows 7 中之虛擬機器管理平台 Workstation 12 內，新增 vSphere 6 虛擬機作業平台，最後建置試驗機器 Ubuntu-desktop 14.04，此部份為巢狀虛擬平台為試驗主軸。

1. 比較對象：B 架構

目的：巢狀虛擬與單一虛擬機管理平台（Workstation 12）之效能差異。

2. 比較對象：C 架構

目的：主機型架構（Workstation 12）與裸機架構（ESXi 6）之效能差異。

3. 比較對象：D 架構

目的：巢狀虛擬化與無虛擬化環境之效能差異。

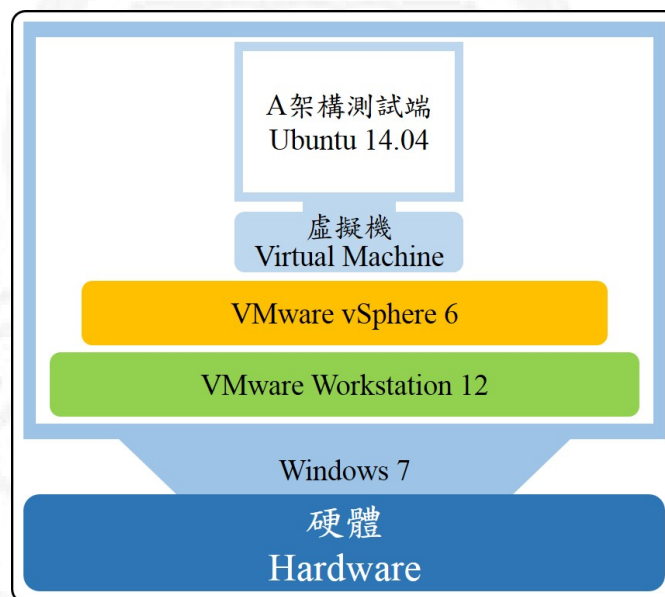


FIGURE 3.2: A 架構示意圖

B 架構設置：於 Windows 7 中之虛擬機器管理平台 Workstation 12 內，建置試驗機器 Ubuntu-desktop 14.04。

1. 比較對象：D 架構

目的：Workstation 12 與無虛擬化之效能差異。

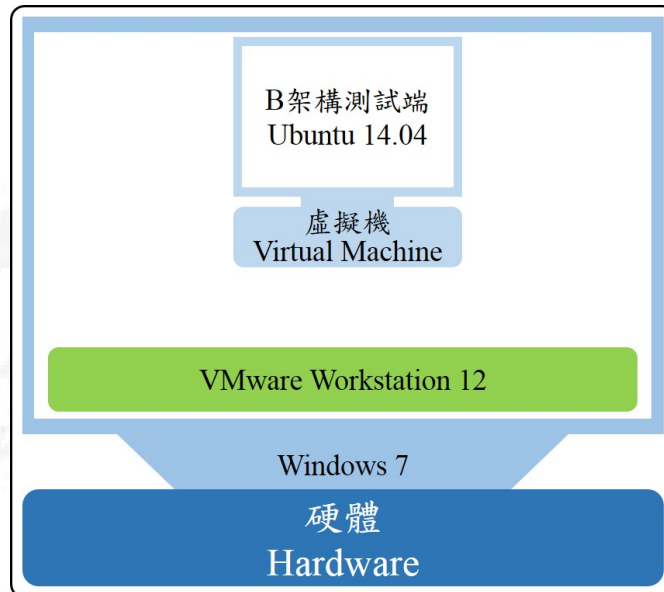


FIGURE 3.3: B 架構示意圖

C 架構設置：於 ESXi 6 中，建置試驗機器 Ubuntu-desktop 14.04，此部份為 ESXi 6 裸機虛擬平台為試驗主軸。

1. 比較對象：A 架構

目的：巢狀虛擬化與裸機架構之效能差異。

2. 比較對象：D 架構

目的：裸機架構（ESXi 6）與無虛擬化環境之效能差異。

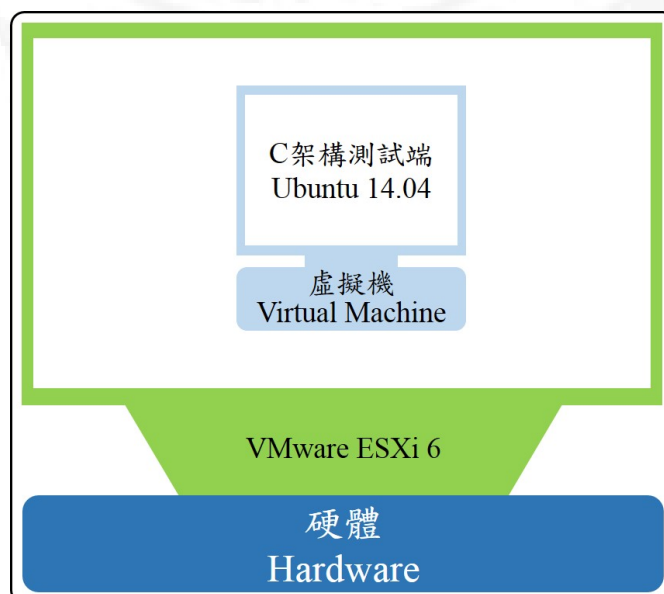


FIGURE 3.4: C 架構示意圖

D 架構設置：於 Ubuntu-desktop 14.04 直接進行測試，此部份無使用虛擬機管理平台，其試驗主軸為取得基礎效能作為實驗對照組。

1. 比較對象：A.B 架構

目的：巢狀虛擬化與單一虛擬機管理平台（Workstation 12）、無虛擬化之效能差異。

2. 比較對象：B.C 架構

目的：主機型架構（Workstation 12）與裸機架構（ESXi 6），以及無虛擬化環境之效能差異。

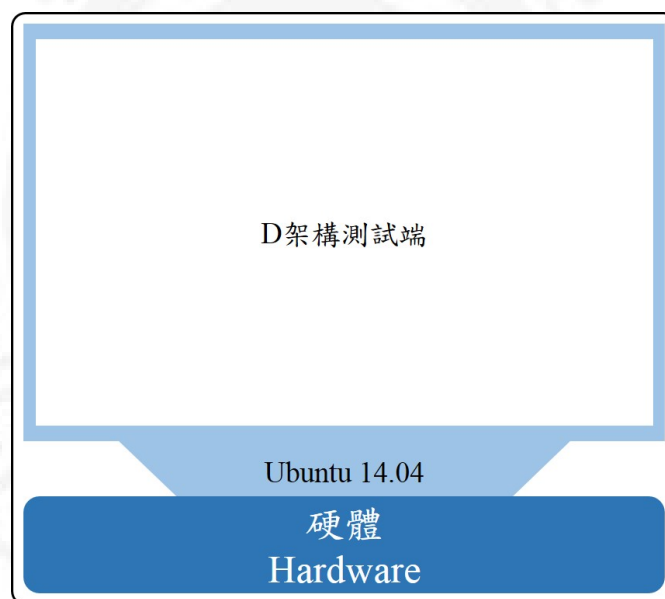


FIGURE 3.5: D 架構示意圖

E 架構設置：比較出最佳效能表現之主機，安裝 KVM 作為虛擬化平台，並取出底層虛擬機實驗數據。

1. 比較對象：B 架構

目的：比較基於主機型架構，且不同作業系統層下之不同虛擬化平台之效能。

2. 比較對象：C 架構

目的：比較主機型架構與裸機架構的效能差異。

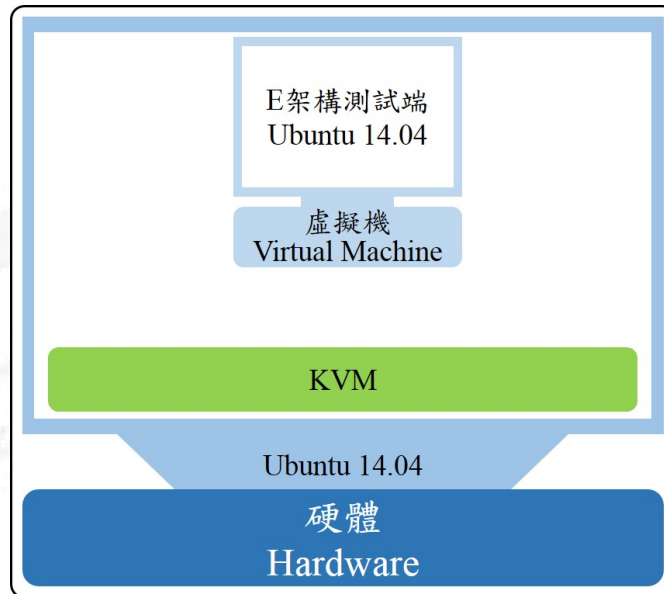


FIGURE 3.6: E 架構示意圖

## 3.2 機器規格

實驗開始前，我們使用二台實體 Intel 主機，其 E3-1230 主要硬體規格四核心處理器（支援 AVX 與 FMA3 技術），記憶體 32GB；i7-3960X 六核心處理器，記憶體 48GB；建議硬碟容量最好高於 300GB 以上，以符合於實驗架構下之最終測試機所需之預設規格 100GB。

然後，依照虛擬化平台狀況，來將實驗的四個實驗架構分別部署於 3 顆各 1TB 硬碟中，下表為詳細硬體規格。

TABLE 3.4: E3 硬體規格

英文名稱	中文名稱	型號 / 規格	數量
Mainboard	主機板	ASUS TS110-E8-PI4	1 片
Power Supply	電源供應器	PCA022 Rev : C/DC300W	1 顆
CPU	處理器	Intel Xeon E3-1230 V3 3.30GHz	1 顆
RAM	記憶體	Transcend DDR3-1600MHz 8Gx4 片	32G
HD	硬碟	Seagate HDD3.5" 1TB/7200RPM	3 顆

單位：單機/台



TABLE 3.5: i7 硬體規格

英文名稱	中文名稱	型號 / 規格	數量
Mainboard	主機板	ASUS P9X79	1 片
Power Supply	電源供應器	RS-400-ACAA-D3 DC400W	1 顆
CPU	處理器	Intel Core i7-3960X 3.30GHz	1 顆
RAM	記憶體	Transcend DDR3-1333U 8G/4G 各 4 片	48G
HD	硬碟	Seagate HDD3.5" 1TB/7200RPM	3 顆

單位：單機/台

### 3.2.1 主機板

E3 主機板規格為 ASUS TS110-E8-PI4，支援 Intel Xeon Processor E3-1230 v3 處理器，記憶體插槽總數 =4 (2 通道) ECC，最大記憶體容量至 32GB；資料儲存應用配額 SATA3 6Gb/s 連接埠 x2，SATA2 3Gb/s 連接埠 x4。

i7 主機板規格為 ASUS P9X79，適用 Intel Core i7 系列處理器，記憶體插槽總數 =8 非 ECC，最大記憶體容量至 64GB；資料儲存應用配額 SATA 6Gb/s 連接埠 x2，SATA 3Gb/s 連接埠 x4。

### 3.2.2 中央處理器

中央處理器 CPU (Central Processing Unit) 為一電腦運作的核心，主要由控制單元與邏輯/算數單元組成，進行資料處理以及運算。

- 一、邏輯/算術單元：資料之邏輯判斷與算術。
- 二、控制單元：控制與協調各單元之間的運作。
- 三、暫存器：CPU 內部用來暫時存放資料的地方。

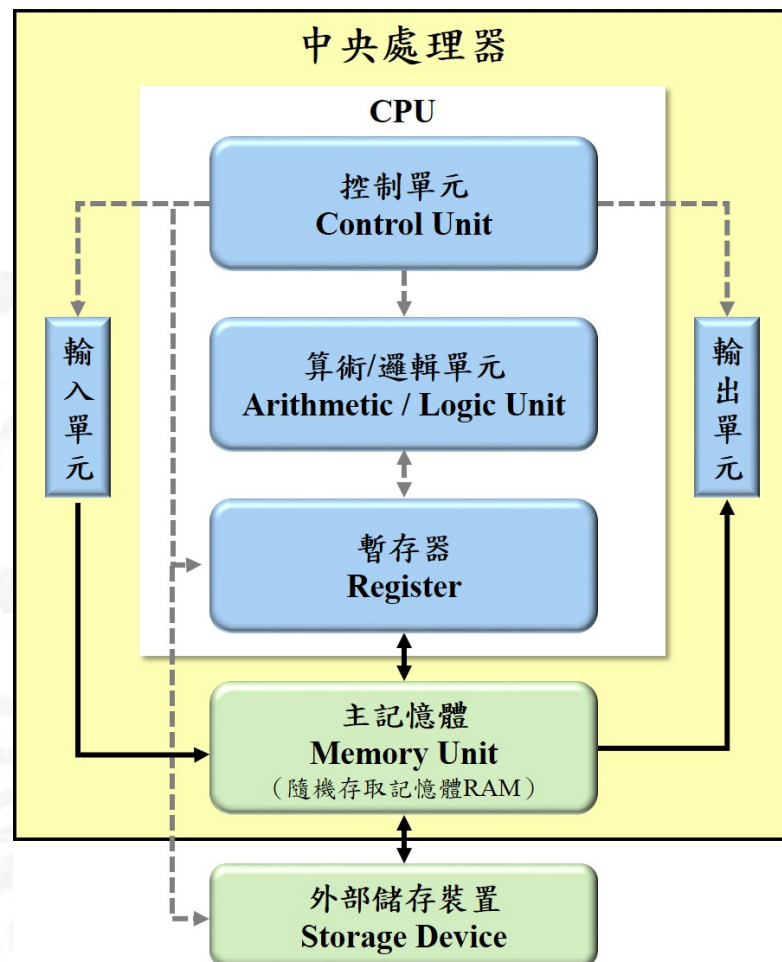


FIGURE 3.7: 中央處理器構成

CPU 的運算能力與處理器基礎頻率（時脈 Clock）…等，都會影響到電腦的執行速度，時脈指的是 CPU 內部所有單元中每秒所能處理的指令數量，所以通常時脈愈高，數字愈大，執行的速度就愈快。

Intel® Xeon Processor E3-1230 v3 四核心（64 位元）處理器內置 8MB 快取記憶體，指令集擴充 Haswell AVX2 / FMA3，處理器基礎頻率（時脈）3.3GHz，最大超頻 3.7 GHz，支援 AVX 2 與 FMA3 指令集。

Intel® i7-3960X 六核心（64 位元）處理器基礎頻率（時脈）3.3GHz，最大超頻 3.9 GHz，支援 Sandy Bridge AVX 指令集。

Haswell(AVX2、FMA3)、Sandy Bridge(AVX) 指令集能提高浮點運算能力，其倍率對照表如附錄 G。

TABLE 3.6: Intel E3 處理器規格

CPU 型號：Intel Xeon E3-1230 V3 3.30GHz(64 位元)

英文	中文	規格
Core	核心數量	4
Flops	浮點數運算 (位元組)	16(支援 Haswell AVX2 / FMA3)
Freq	處理器基準頻率	3.3 GHz
Max Freq	最大頻率	3.7 GHz

單位：1 顆

TABLE 3.7: Intel i7 處理器規格

CPU 型號：Intel Core i7-3960X 3.30GHz(64 位元)

英文	中文	規格
Core	核心數量	6
Flops	浮點數運算 (位元組)	8(支援 Sandy Bridge AVX 指令集)
Freq	處理器基準頻率	3.3 GHz
Max Freq	最大頻率	3.9 GHz

單位：1 顆

本文中為了能更精準與統一，因四核心 CPU 如使用 Hyper-Threading 技術，電腦將會辨識為 8 顆 CPU，故前置作業需先至 BIOS 中將超執行緒 (Hyper-Threading, HT) 技術設為「禁用」。

### 3.2.3 硬碟

硬碟 HD (Hard Disk) 具有永久性儲存能力之電腦元件，影響電腦整體效能之內部原因可能有本身的磁碟轉速快慢與虛擬記憶體 (swap) 之使用量有關，所謂轉速 (PRM) 為磁頭在磁碟上來回移動到達正確磁軌後，然後磁碟會旋轉到相對應正確的磁區中所花費的時間，當轉速愈高所需時間就愈短；虛擬記憶體 (swap) 是使用硬碟空間充作為實體記憶體的延伸，當記憶體容量配置不足時，系統將硬碟中的一部份來使用，造成硬碟去頻繁的進行 swap 交換動作，當硬碟負擔加重，自然也就會影響系統的效能。不僅如此，其外部原因還有傳輸介面會稍微到影響傳輸速率。本文中硬碟使用了 SATAIII 匯流排介面，傳輸資料採串列方式與具有 6 /Gbps 的理論傳輸率。

### 3.2.4 隨機存取記憶體

隨機存取記憶體 RAM (Random Access Memory) 的部份則需經過下列公式計算，方可符合效能測試的需求。在實驗環境中 Intel Linpack 有要求記憶體 16GB，但如無使用 Intel Linpack 時，其網格運算的計算記憶體需求公式如下，再加上所使用的虛擬化管理平台與作業系統的記憶體使用量，我們計算出實驗所需的記憶體分配。

TABLE 3.8: 網格運算記憶體需求計算公式

網格運算需求 = (網格行數 x 網格列數 x 位元組) /  $10^9$  位元組資料  
 註：一個位元組由 8 個位元組成，為電腦記憶體的基本儲存單位。

TABLE 3.9: 網格運算記憶體需求試算

網格尺寸	Bytes	GB/s (Giga)
行數 x 列數	位元組	每秒傳送 $10^9$ 位元組資料
10000 x 10000	8	0.8GB/s
20000 x 20000	8	3.2GB/s
30000 x 30000	8	7.2GB/s

TABLE 3.10: 實驗架構記憶體需求總量計算公式

需求總量 = 作業系統需求 + 虛擬化平台需求 + 網格計算需求

TABLE 3.11: 實驗所需的記憶體需求預估

作業系統與平台	架構 A	架構 B	架構 C	架構 D	架構 E
Windows 7	3	3	•	•	•
Workstation 12	1.5	1.5	•	•	•
vSphere 6 / ESXi 6	1.5	•	1.5	•	•
Ubuntu14.04	2	2	2	2	2x2
KVM	•	•	•	•	1
Intel Linpack	16	16	16	16	16
預估需求總量	24	22.5	19.5	18	21

單位：GB

註：實驗項目為依序進行，故以最高使用記憶體量的測試項目「Intel Linpack」計。

由上圖來看，實驗主機之 32GB 與 48GB 記憶體是充足的；故經評估之後，最終測試機 Ubuntu 給定 24GB 記憶體分配量，因測試環境中之記憶體容量配置不足時，系統會自動使用硬碟的一部份空間作為虛擬記憶體（swap）使用，此部份會產生系統效能下降和增加 CPU 的負荷等各項延遲，故預設給定的記憶體數量需充足。

### 3.3 系統與軟體版本

作業系統（operating system，OS）為管理電腦軟硬體資源的電腦程式，也是系統中的核心，該系統中所需要管理如記憶體配置、資源分配，輸出入裝置、網路介面設定，檔案系統管理等事務執行操作，亦提供使用者與系統互動的操作介面。

一個標準作業系統應該提供以下的功能：

- 一、驅動程式（Device drivers）
- 二、處理執行管理（Processing management）
- 三、記憶體管理（Memory management）
- 四、檔案系統（File system）
- 五、使用者介面（User interface）
- 六、網路通訊（Networking）
- 七、安全機制（Security）

#### 3.3.1 系統版本

##### 3.3.1.1 Windows

由 Microsoft 所發行之 Windows 7 作業系統，搭配多核心處理器使用可以充份滿足使用者與系統管理者的需求，所有 32 位元版本的 Windows 7 最多都

可以支援 32 個處理器核心，而 64 位元版本則最多可以支援 256 個處理器核心。



FIGURE 3.8: Windows 畫面-E3



FIGURE 3.9: Windows 畫面-i7

### 3.3.1.2 Ubuntu

Ubuntu 由馬克·舍特爾沃斯創立，以 Debian 為開發藍本的開源作業系統，適用多數自由開源軟體，但需留意欲安裝之軟體與硬體及其版本的相容性，該軟體更新週期約為 6 個月，14.04 桌面版與伺服器版都有 5 年長期支援版本 (Long Term Support, LTS) 桌面版 3 年，伺服器版 5 年。



FIGURE 3.10: Ubuntu 畫面-E3



FIGURE 3.11: Ubuntu 畫面-i7

### 3.3.2 虛擬化平台版本

虛擬化 (Virtualization) 為將實體資源中之硬碟、記憶體、網路等作抽象、轉換的資源管理技術，使用者可以以更好的方式加以應用這些資源，且實體機器硬體與虛擬機器內運行的作業系統之間，會有軟體介面來控制與隔離虛擬機器對於實體機器中的硬體存取。

為確保主機環境與各項硬體的一致性，使效能測試更趨精準，作業系統安裝前需先進入基本輸出輸入系統 (BIOS) 設定，使用硬碟開機，並確認各項所需之設定是否完整 (例：Intel virtualization Technology [ Enabled ])；在各自虛擬機建立時皆不使用快照功能來建立虛擬機，以求系統之單一、完整性；安裝完作業系統後需連線網際網路來進行更新，並確定各測試機皆處於相同預設狀態；執行效能測試前，為確保不被其他應用程式佔用資源，可先將一些不必要的功能關閉，以及關閉部份不需使用的應用程式，以達到測試求出之數據更趨精準。

測試機預設基礎規格要求：記憶體 24G、硬碟容量 100GB、開啟虛擬化功能，移除不必要之硬體安裝 (例：印表機、音效卡、顯示器不需使用 3D 加速)，可視需求狀況安裝 USB 控制器。



### 3.3.2.1 VMware Workstation

VMware Workstation 12 是管理虛擬機器 (Virtual Machine) 平台，可於目前所使用的作業系統中新增多部不同作業系統的虛擬機器，並自訂磁碟分割及各項硬體設備，在現有的資源作最大化的利用，但需留意到安裝時的相容性問題及資源分配狀態；可同時啟動數台虛擬作業系統，系統管理人員可統一集中管理機器及備份。

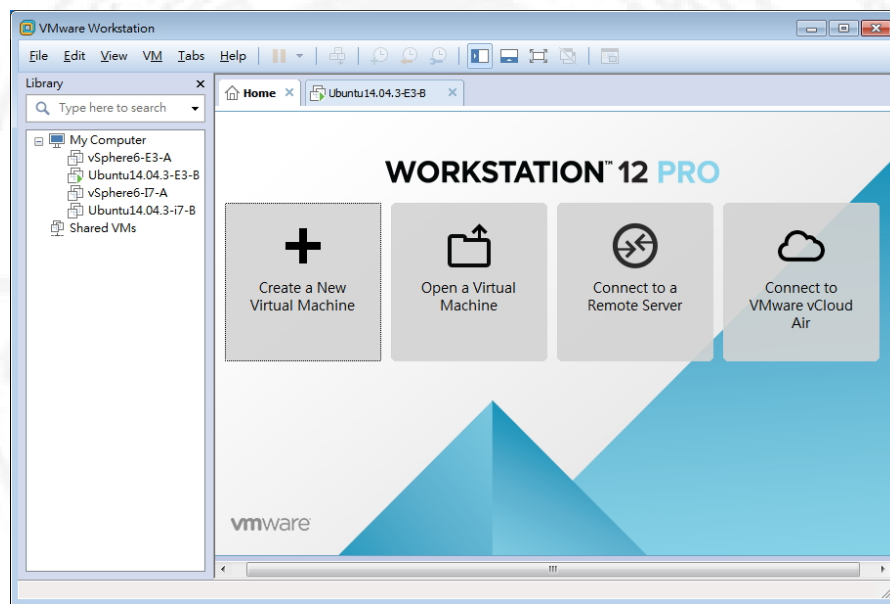


FIGURE 3.12: VMware Workstation 介面

### 3.3.2.2 VMware vSphere / ESXi

VMware vSphere 6 / ESXi 6 提供虛擬資料中心一致的管理，為一伺服器虛擬化平台，可應用於主機型架構下或是直接於裸機下作安裝，管理者可經由 vSphere 6 / ESXi 6 主控端收集、查看系統與使用者目前使用狀態，提供即時效能數據及資源利用率…等功能。

本文中 vSphere 應用於主機型架構，ESXi 則應用於裸機型架構。



FIGURE 3.13: vSphere 介面

### 3.3.2.3 KVM

Kernel Virtual Machine (KVM)，為開源 Linux 虛擬管理平台，KVM 建立虛擬機之上限取決於實體機的資源，所以 KVM 中所有虛擬機之效能影響關鍵在於實體機的資源分配；優勢為啟動虛擬機之速度較 vSphere / ESXi 來的迅速，但主要缺點有：發行於 Linux 版本中，且操作介面的親和度稍差，故管理者或使用者的技術門檻相對提高，但因其為開源軟體，可降低企業虛擬化部署成本。

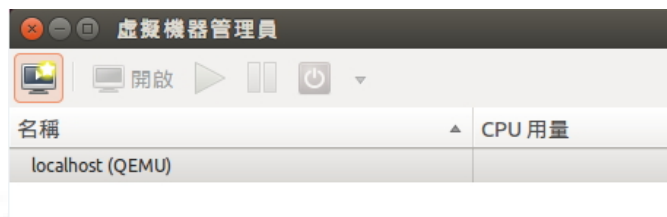


FIGURE 3.14: KVM 介面

# Chapter 4

## 實驗環境與結果

### 4.1 測試架構

#### 4.1.1 基礎測試環境架構

本文中共分為五個架構，最終使用 Ubuntu-desktop 14.04 作業系統進行實測；測試工具為高效能運算測試套件 (HPC Challenge Benchmark Suite,HPCC)、GNU 編譯器 (GNU Compiler Collection,GCC)、Intel C++ 編譯器 (Intel C++ Compiler,ICC)；測試項目為基本線性代數子程序 BLAS (Basic Linear Algebra Subprograms)：GotoBlas、OpenBlas、數學核心函數庫 (Intel Math Kernel Library,MKL)。

#### 4.1.2 測試架構說明

為為求數據準確性高，本文中皆使用相同的軟體版本進行測試，包含 Windows 7、Ubuntu-desktop 14.04、VMware Workstation 12、vSphere / ESXi 6、Intel Parallel Studio XE 2016...等。

A.B.C.D 架構之最終測試作業系統層皆為 Ubuntu-desktop 14.04，於不同編譯器 GCC、ICC 之測試項目則特別依序去修改網格測試值 10,000、20,000、30,000 直至各測試項目完成，再使用 Intel Linpack 作最後的測試項目，於最佳效能表現之主機上安裝部署 E 架構之 KVM 進行測試。

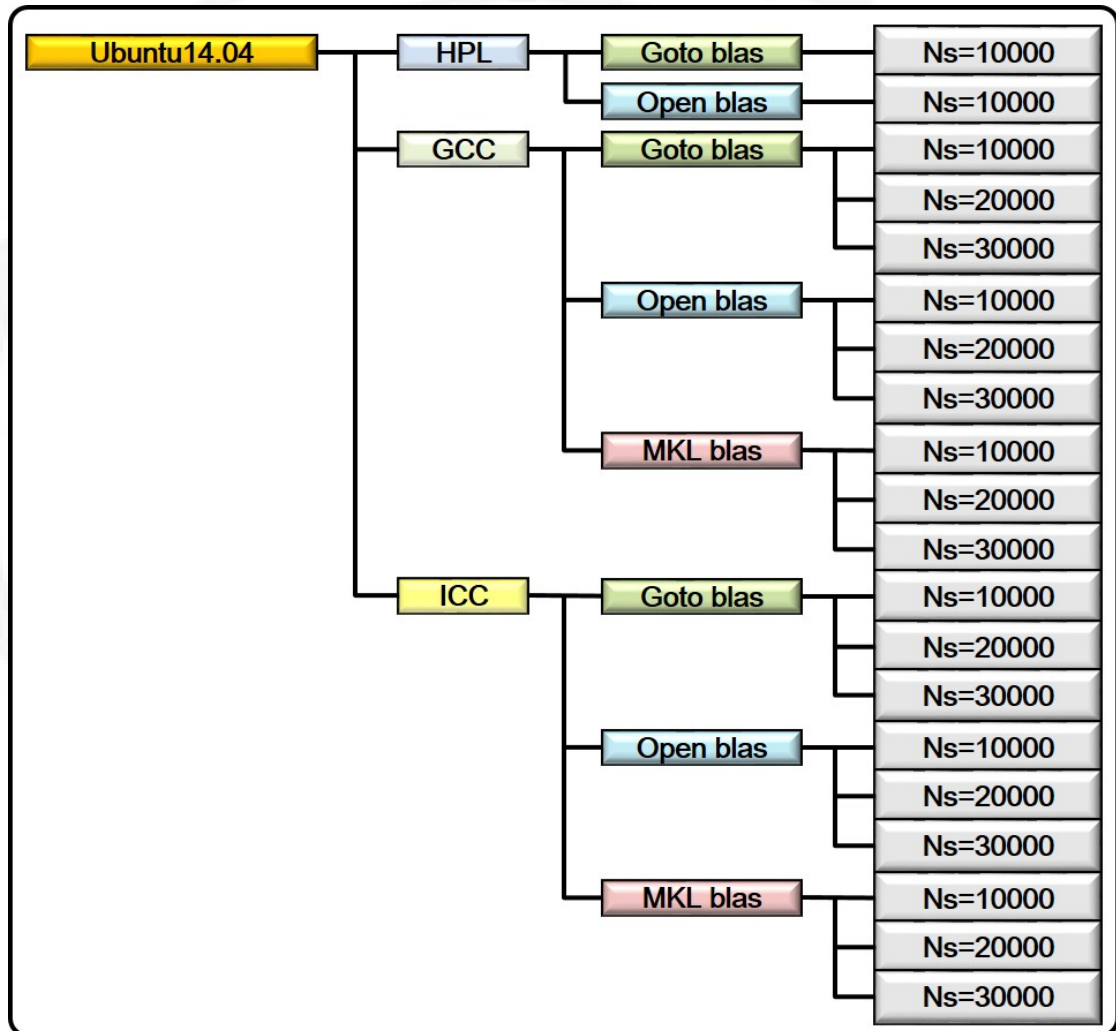


FIGURE 4.1: 實驗項目

需修改之網格設定值及需了解的變數，於下圖作說明。

```

1 HPLinpack benchmark input file #描述此檔案的參數 (不需修改)
2 Innovative Computing Laboratory, University of Tennessee #同上 (不需修改)
3 HPL.out output file name (if any) #輸出檔名稱 (不需修改)
4 8 device out (6=stdout,7=stderr,file) #設定輸出方式 (不需修改)
5 1 # of problems sizes (N) #多少數量的矩陣大小 (不需修改)
6 30000 Ns #矩陣的大小(本實驗將修改此條件，分別為10000,20000,30000) (修改)
7 4 # of Nbs #區塊大小的數量 (不需修改)
8 88 128 108 168 Nbs #矩陣分割區塊大小，又稱分區分塊(通常為256以下，以實測求得最佳值) (不需修改)
9 0 PMAP process mapping (0=Row-,1=Column-major) #Row-major 適用於CPU數較少且節點數較多的系統;Column-major則反之 (不需修改)
10 1 # of process grids (P x Q) #PxQ=總核心數=進程數 (不需修改)
11 2 Ps #進程行數 (不需修改)
12 2 Qs #進程列數 (不需修改)
13 16.0 threshold #測試的精準度 (不需修改)
14 3 # of panel fact #14~21行：表示L分解方式 (不需修改)
15 0 1 2 PFACTs (0=left, 1=Crout, 2=Right)
16 1 # of recursive stopping criterium
17 4 NBHINs (>= 1)
18 1 # of panels in recursion
19 2 NDIVs
20 1 # of recursive panel fact.
21 1 RFACTs (0=left, 1=Crout, 2=Right)
22 1 # of broadcast #22~23行：HPL提供之6種廣播方式。 (不需修改)
23 0 BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM) #0,1適合於高速網路，2,3較慢網路速度適用，將數據切割後再傳送 (不需修改)
24 1 # of lookahead depth #24~25行：Message passing的深度(與機器之配置相關) (不需修改)
25 0 DEPTHs (>=0)
26 2 SWAP (0=bin-exch,1=long,2=mix) #26~27行：表示U的廣播方式 (不需修改)
27 64 swapping threshold
28 0 L1 in (0=transposed,1=no-transposed) form #28行：表示L的存放格式，transposed 為按列存放，反之按行存放 (不需修改)
29 0 U in (0=transposed,1=no-transposed) form #29行：表示U的存放格式，transposed 為按列存放，反之按行存放 (不需修改)
30 1 Equilibration (0=no,1=yes)
31 8 memory alignment in double (> 0) #記憶體對齊方式 (不需修改)

```

FIGURE 4.2: 各項變數定義

## 4.2 理論值計算

為計算出各效能比例，需先計算出基礎 CPU 處理能力理論值，其計算公式與所需變數名稱定義如下：

TABLE 4.1: 理論值計算公式

$$\text{Peak1}(\text{freq} \times \text{core} \times \text{flops})$$

TABLE 4.2: 效能比變數名稱定義

英文	中文及變數解釋
Compile library(min)	浮點數運算能力 (最優值/最小值)
Gflops	浮點數運算結果
eType	浮點數運算單位
Efficiency	效能比
Peak1	頻率週期
freq	處理器基準頻率
core	核心數量
flops	浮點數運算 (位元組)

為求慎重，我們在此提供 Intel 公佈之 E3-1230 V3 與 i7-3960X 之浮點運算數據作為佐證，請見附錄 G。

TABLE 4.3: E3 CPU 理論值計算

CPU : Intel Xeon E3-1230 V3 3.30GHz(AVX2、FMA3)						
中文	基準頻率	頻率上限	核心數	浮點數	基準值	基準值 (最高)
英文	Freq	Max Freq	Core	Flops	Peak1	Peak2
計算值	3.3	3.7	4	16	211.2	236.8

TABLE 4.4: i7 CPU 理論值計算

CPU : Intel Core i7-3960X 3.30GHz(Haswell AVX)						
中文	基準頻率	頻率上限	核心數	浮點數	基準值	基準值 (最高)
英文	Freq	Max Freq	Core	Flops	Peak1	Peak2
計算值	3.3	3.9	6	8	158.4	187.2

TABLE 4.5: 效能與理論值計算比例公式

$$\text{Efficiency} = \text{Compile library}(\text{Gflops} \times \text{eType}) / \text{Peak1}(\text{freq} \times \text{core} \times \text{flops})$$

## 4.3 實驗方法

### 4.3.1 HPL 安裝與實測

首先我們完成各架構所需之虛擬機管理平台與作業系統的配置，同時也將各項初始化設定完畢，接下來開始在各架構下之主要測試層 Ubuntu-desktop 14.04 作業系統進行安裝 GCC (GNU 編譯器) 以及 HPL 高效能計算程式下載安裝與設定，因 Ubuntu-desktop 14.04 原始系統環境中已包含 GotoBlas、OpenBlas 二種高性能多核心 Blas 庫，故安裝後可先進行 HPL 測試效能，並取得各架構效能基準值，完成 HPL 測試後，其它測試項目皆需進行變更網格大小續測，直至各架構之實驗機均完成測試項目。

### 4.3.2 GotoBlas 與 OpenBlas 測試

此部份特別需要留意的是我們後續的實驗將關係到網格運算 (Grid computing)，其用意在於系統中的作業分配，網格變數我們預設為 N 值，本文中將變更其 N 值為 10,000、20,000、30,000，實驗後得知當作業分配的愈趨細緻時，效能愈佳，但所需執行的時間將明顯增加。

#### 4.3.2.1 GCC 編譯器執行測試

使用 GCC 編譯器接續進行 GotoBlas 與 OpenBlas 之實測，並取其結果數據作為評估。

#### 4.3.2.2 GCC 執行 Intel MKL 測試

本測試前需先進行 ICC 編譯器安裝（請見下一小節），並引入環境變數。

使用 GCC 編譯器接續進行 Intel MKL 之實測，並取其結果數據作為評估。

#### 4.3.2.3 ICC 編譯器安裝與測試

實驗進行前先進行 ICC 編譯器安裝與執行（Intel Parallel Studio XE 2016），請先至 Intel 官網申請下載，需輸入個人資料並至您留下的聯絡信箱內讀取授權碼。<https://software.intel.com/en-us/intel-parallel-studio-xe>

使用 ICC 編譯器接續進行 GotoBlas 與 OpenBlas 之實測，並取其結果數據作為評估。

#### 4.3.2.4 ICC 執行 Intel MKL 測試

使用 ICC 編譯器接續進行 Intel MKL 之實測，並取其結果數據作為評估。

#### 4.3.2.5 Intel Linpack 測試

使用 Intel® Parallel Studio XE 2016 安裝時所搭載之 Linpack 再次進行測試，並以其測試結果與 HPL 所測出之數據進行比對。



#### 4.3.2.6 實驗加測組

本論文中 E 架構為實驗加測組，於二主機各四架構之各項測試完成後，經由比較主機的整體效能最佳者，再增加測試 E 架構，並取其結果數據作為評估。

#### 4.3.2.7 數據彙整

將已取得之效能資訊收集與數據彙整後（執行效能、分區分塊因子與不同網格大小設定表現…等），取出各組數據表現最佳值，開始進行分析比對，並應用統計分析圖表，利於觀察其差異性，看出各組表現與整體評估，所得數據與分析結果經總結後發表於下一節。

## 4.4 實驗結果

### 4.4.1 HPL 實測結果

E3、i7 二主機於各架構的 HPL 測試 Open Blas、Goto Blas 相比，Open Blas 較佳。

由 A 架構巢狀虛擬與 B 架構單一虛擬平台，比較巢狀虛擬是否影響效能：

1. 於 E3 主機，於 Goto Blas：B 架構略優於 A 架構；於 OpenBlas：A 架構略優於 B 架構。
2. 於 i7 主機，於 Goto Blas：B 架構略優於 A 架構；於 OpenBlas：B 架構略優於 A 架構。

因 E3 主機無法真正確認出巢狀虛擬是否帶來影響，故我們可由 4.4.5 Intel Linpack 實測來進行確認。

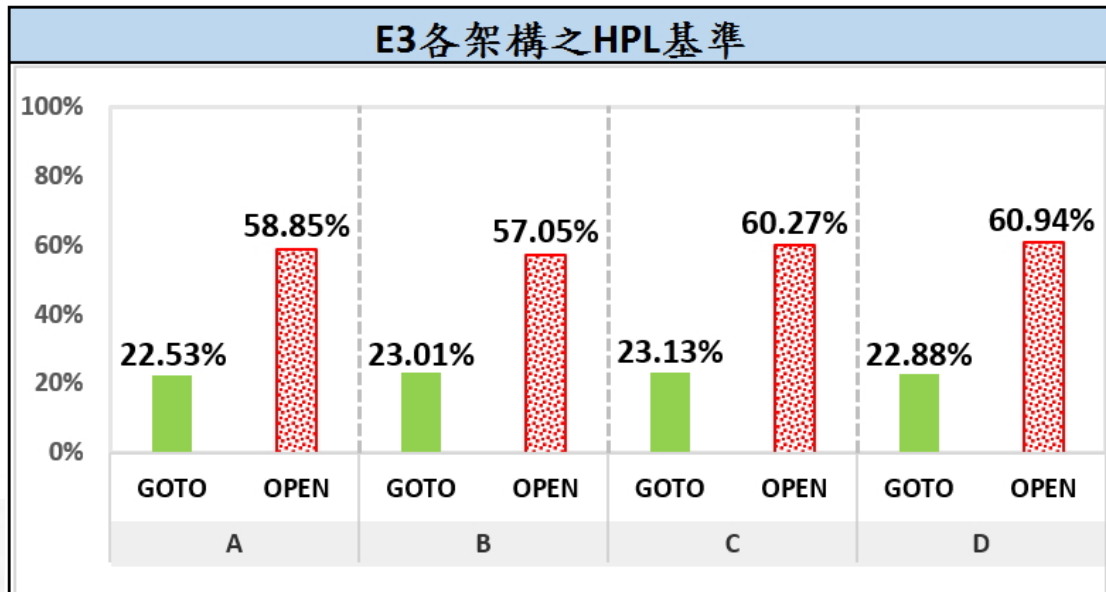


FIGURE 4.3: HPL 實測結果呈現圖-E3

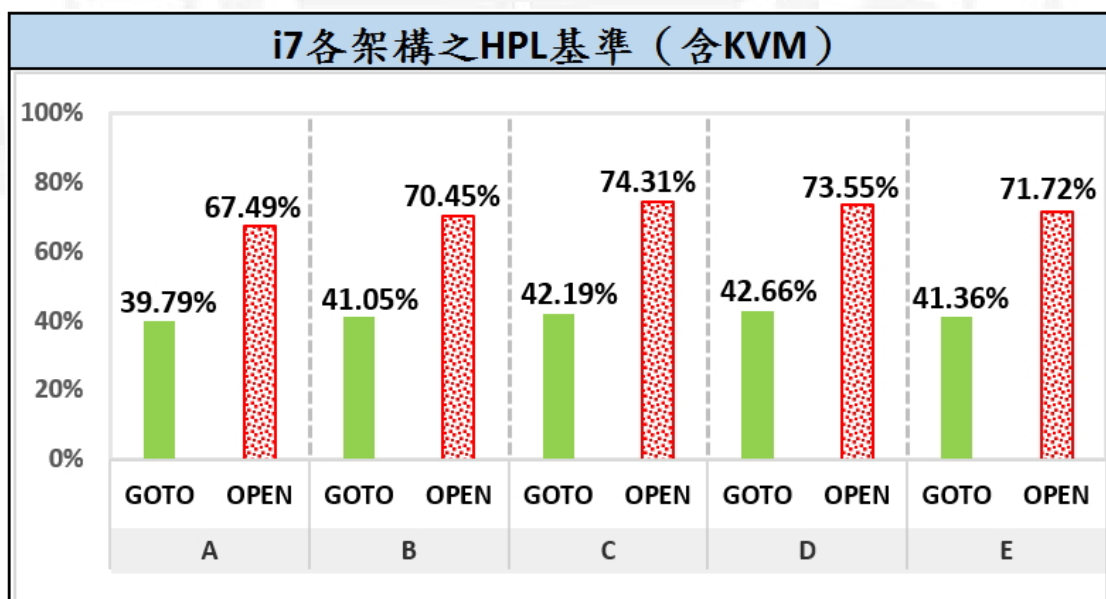


FIGURE 4.4: HPL 實測結果呈現圖-i7

## 4.4.2 GCC 編譯器實測結果

E3、i7 二主機於 C、D 架構效能較佳，且網格切分愈細緻，效能表現愈佳，但所需時間愈長。

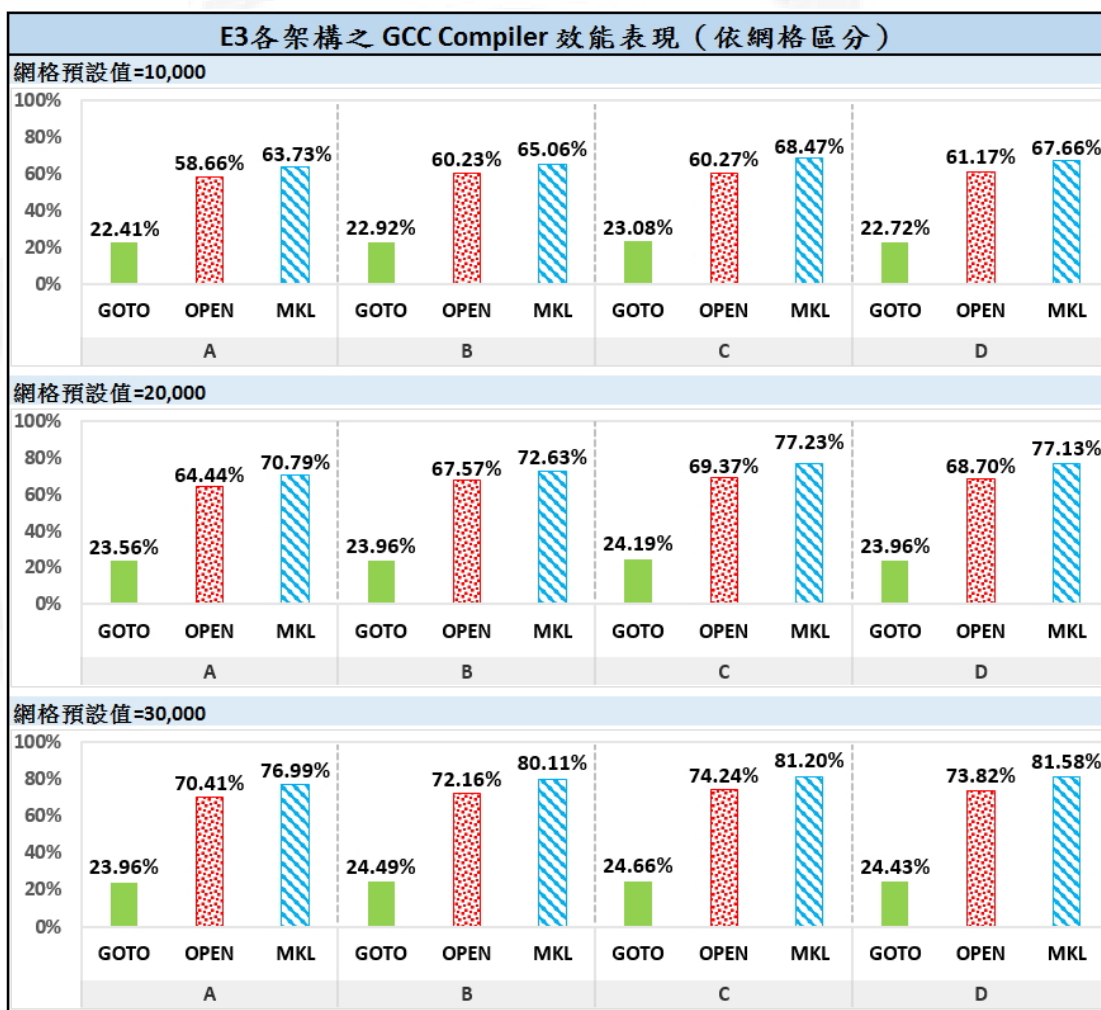


FIGURE 4.5: GCC 編譯器實測結果呈現圖-E3

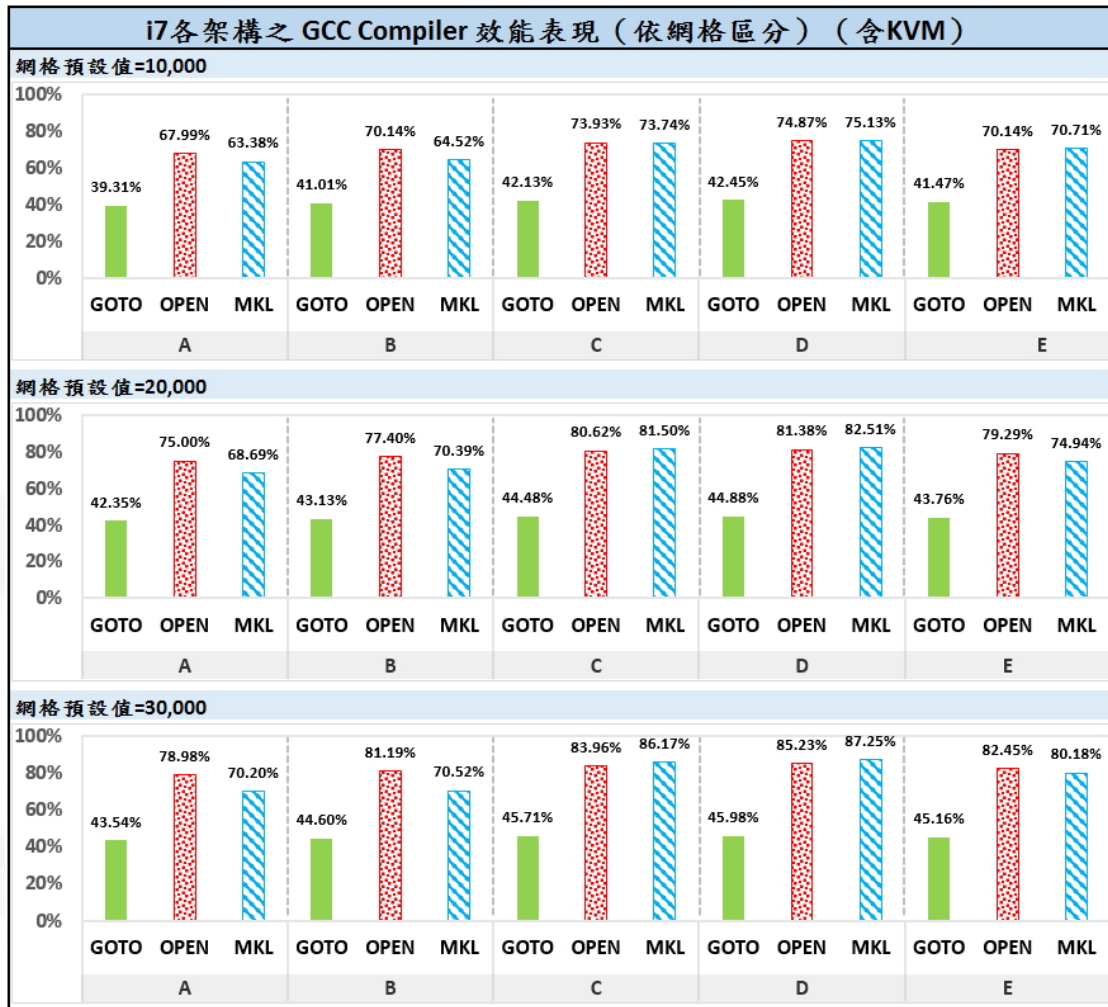


FIGURE 4.6: GCC 編譯器實測結果呈現圖-i7

## 4.4.3 ICC 編譯器實測結果

E3、i7 二主機於 C、D 架構效能較佳，且網格切分愈細緻，效能表現愈佳，但所需時間愈長。(與 GCC 編譯結果相近)。

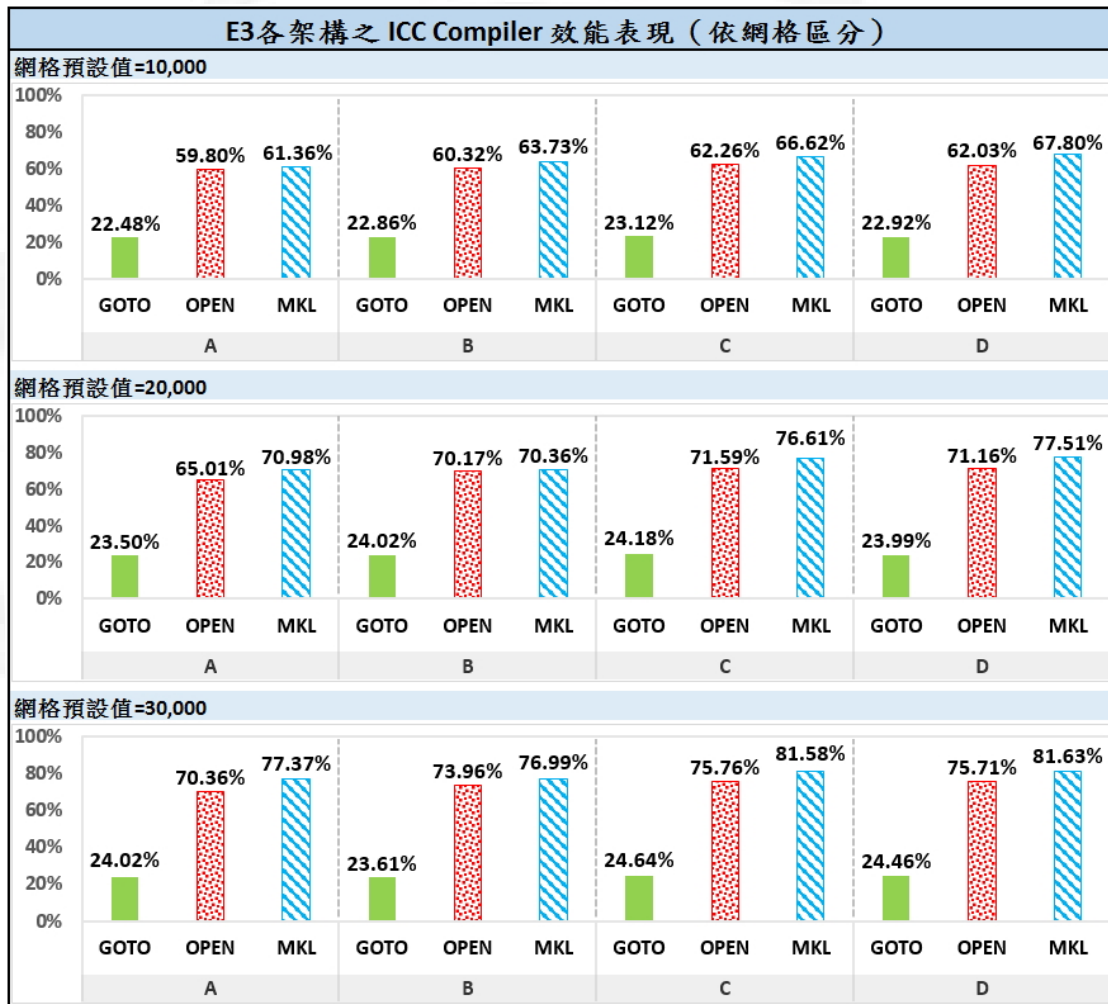


FIGURE 4.7: ICC 編譯器實測結果呈現圖-E3

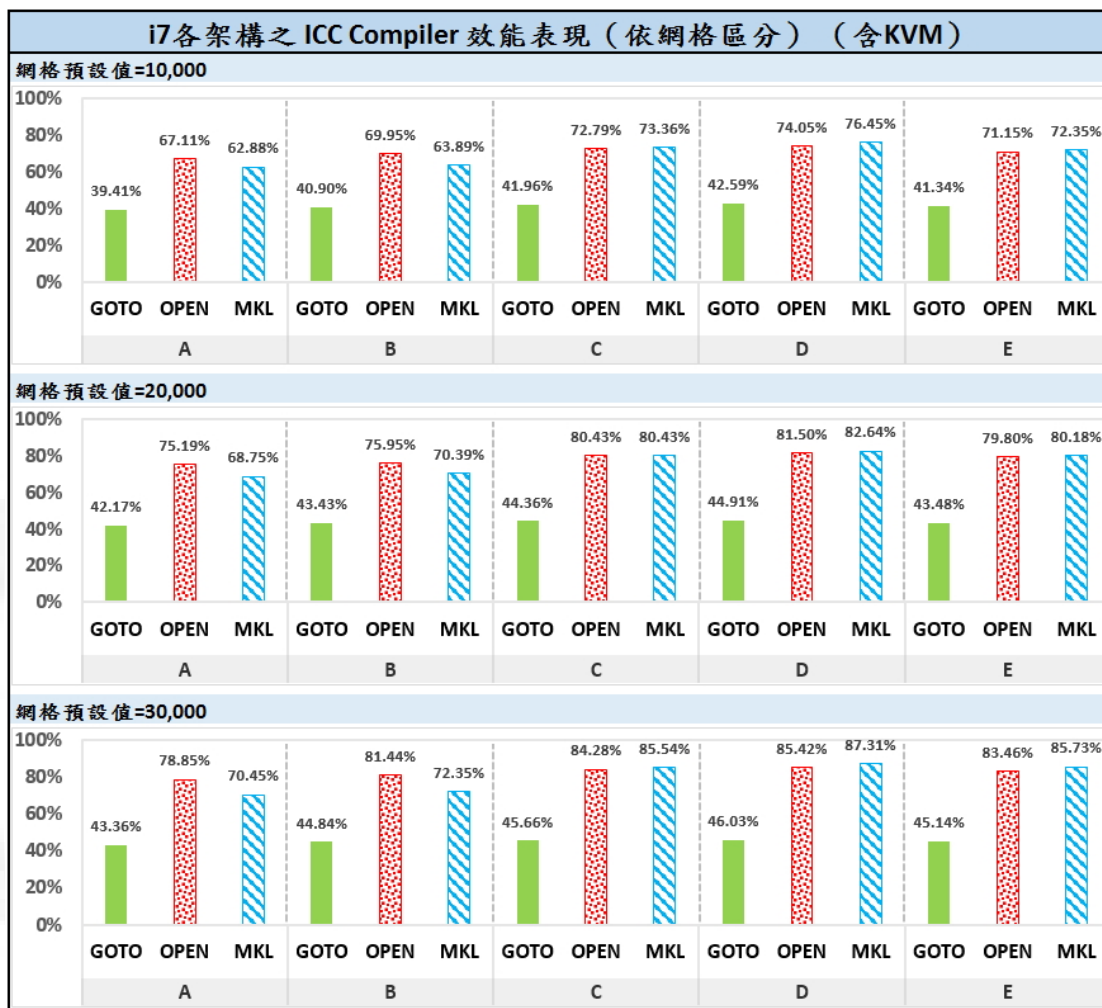


FIGURE 4.8: ICC 編譯器實測結果呈現圖-i7

## 4.4.4 MKL 數學核心函式庫實測結果

E3、i7 二主機於 C、D 架構的效能表現較優於其他架構。

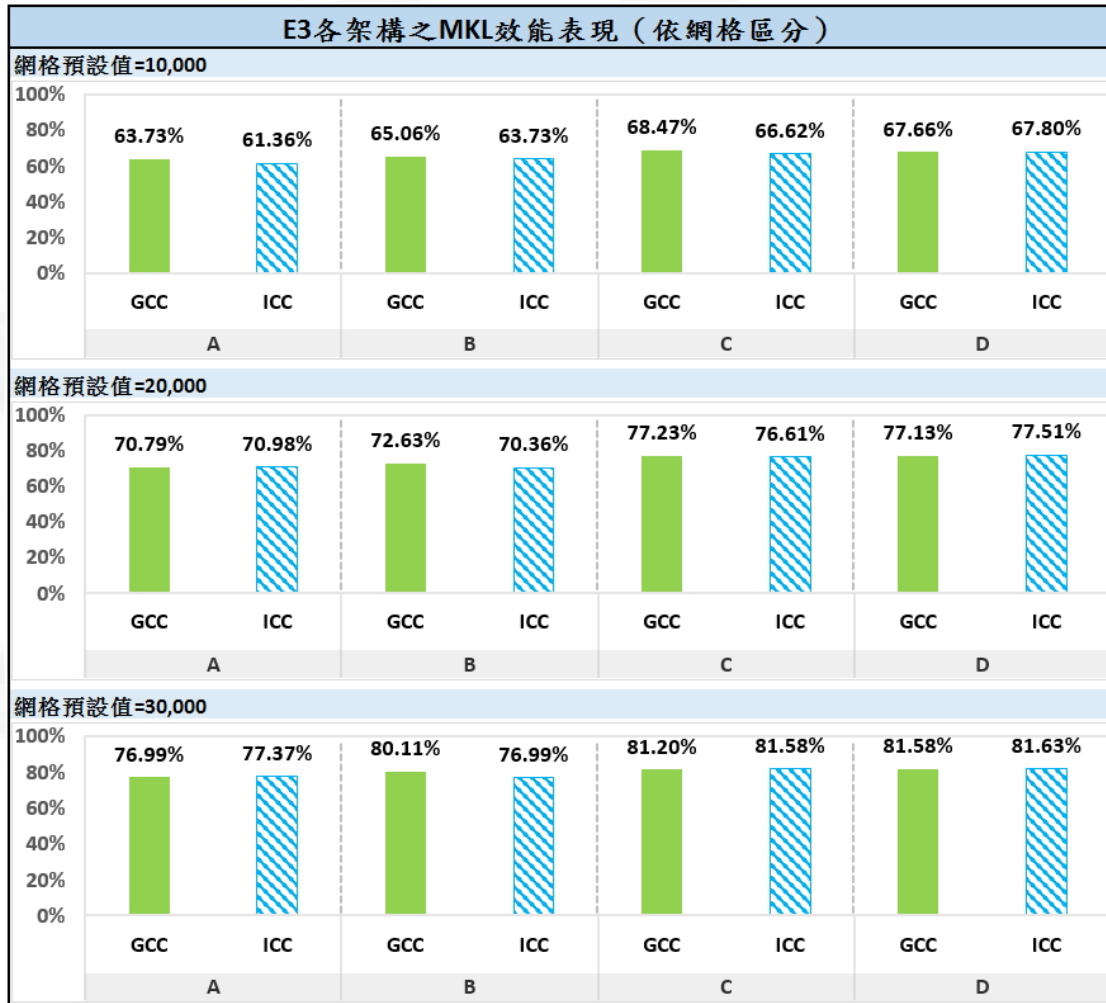


FIGURE 4.9: MKL 實測結果呈現圖-E3

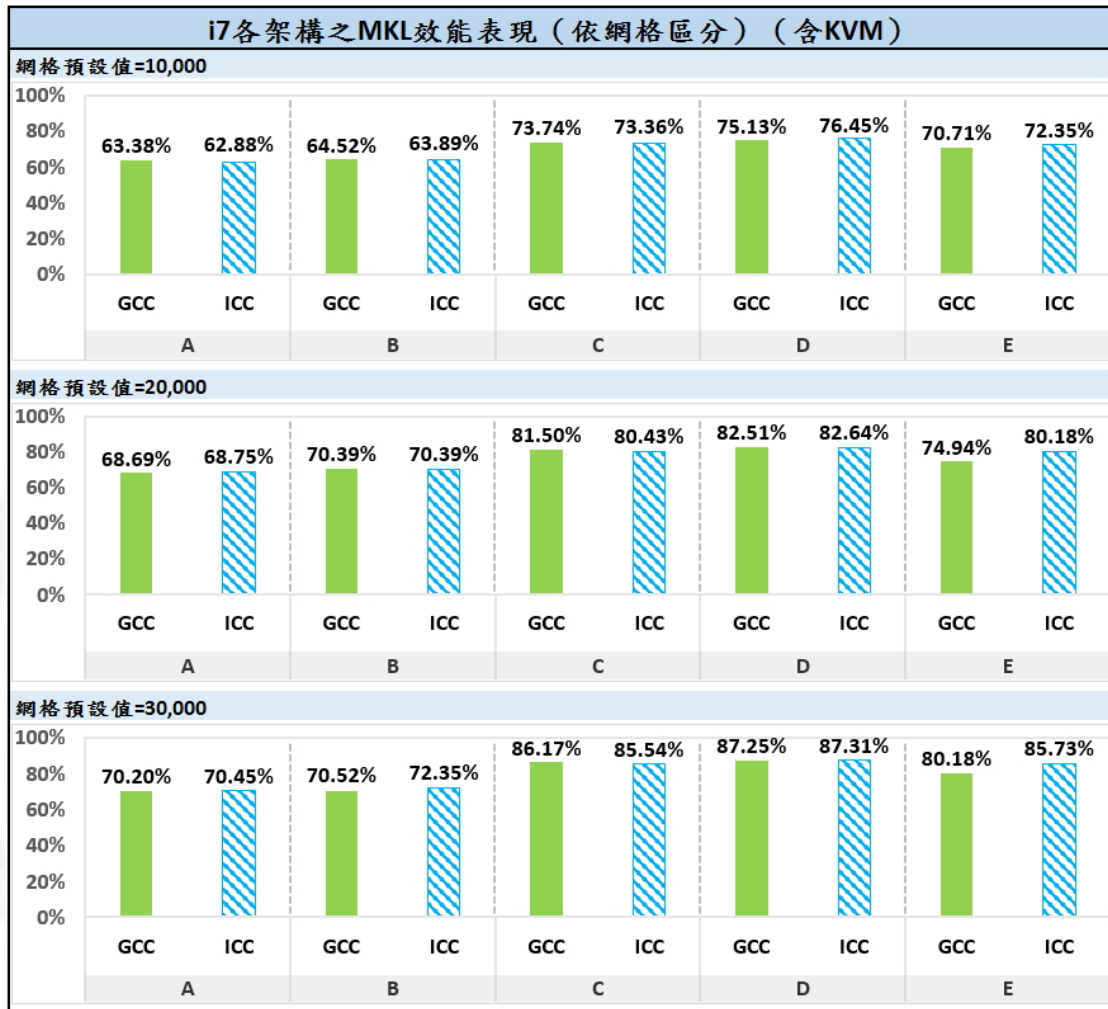


FIGURE 4.10: MKL 實測結果呈現圖-i7



#### 4.4.5 Intel Linpack 實測結果

此部份測試需符合測試工具需求（該工具預設所需記憶體為 16G 以上，方能進行測試），我們在本文中是使用至 24G 的記憶體，故此部份不影響測試進行。

Intel Linpack 基準測試下以網格預設 30,000 的情況下來看，以 C、D 架構較佳；而 A、B 架構的組成是於 Windows 系統底下，再使用 VMware Workstation 與 vSphere 之巢狀虛擬化，由此可知巢狀虛擬化之階層多寡將影響虛擬機的效能表現。

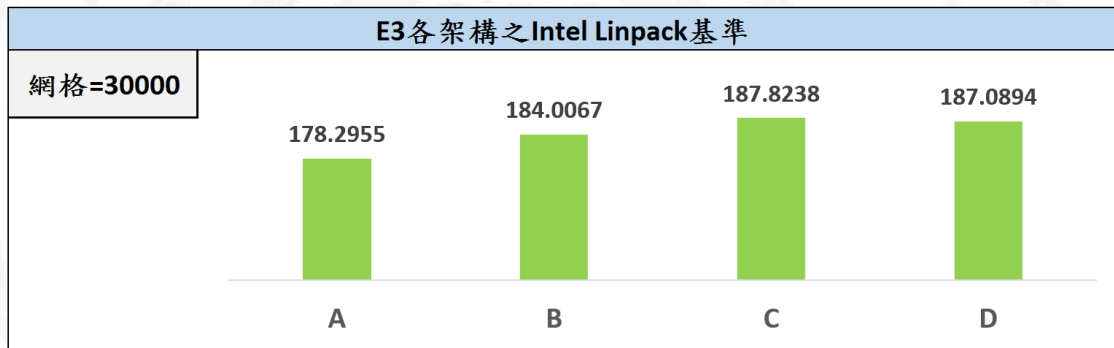


FIGURE 4.11: Intel Linpack 實測結果呈現圖-E3

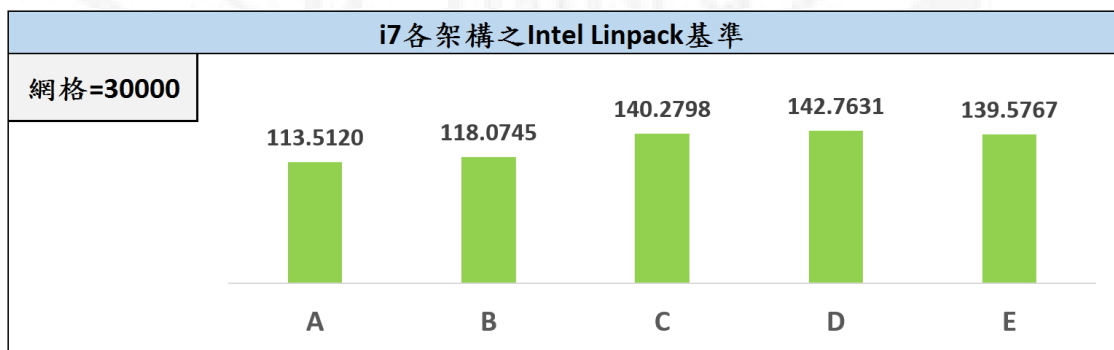


FIGURE 4.12: Intel Linpack 實測結果呈現圖-i7

#### 4.4.6 KVM 實測結果

上述測試已確定 i7 主機效能較 E3 主機來的佳，故於 i7 主機中，再進行 E 架構 KVM 之測試。結果可看出 ESXi 效能仍比 KVM 來的好，基於成本考量，KVM 架構為完全開放源所組成，其效能亦優於 A.B 架構，故 KVM 可作為企業 IT 人員於建立虛擬機的第二種選擇。

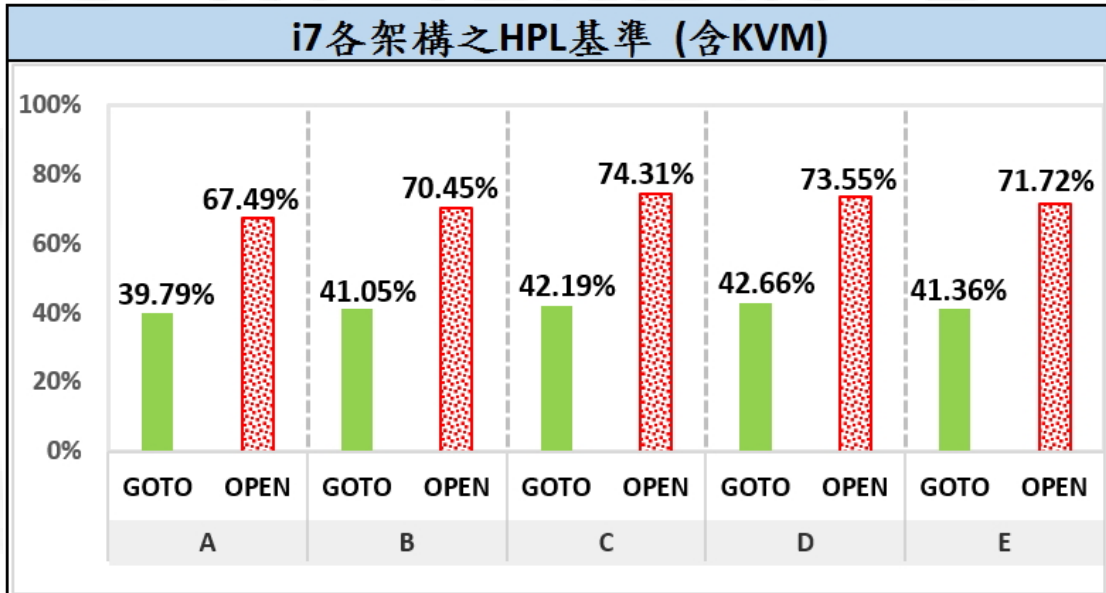


FIGURE 4.13: KVM 實測 HPL 結果-i7

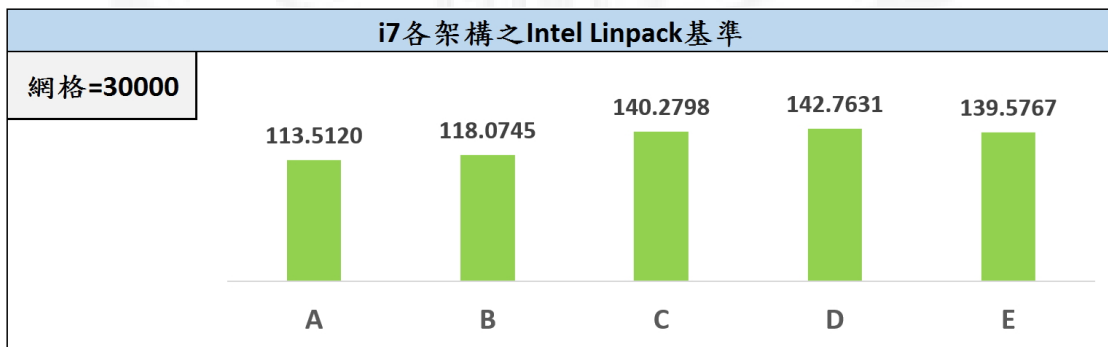


FIGURE 4.14: KVM 實測 Intel Linpack 結果-i7

#### 4.4.7 分區分塊因子分析結果

此部份僅限於本論文中的實驗機器為準，如以不同機器，則仍需進行實際測試方能確定最佳分區分塊因子設定。

於各主要測試項目（除了 Intel Linpack 不計入），每一主機的四個架構中總共取出高達 960 筆數據資料，取出最佳值總數為 80 筆；以架構 A 為例，其測試項目之分區分塊因子（88、108、128、168）中取出最佳的一個尺寸大小，乘以所進行的網格測試值（10,000、20,000、30,000）分類次數，HPL-Goto 網格測試值為 10,000，取出 1 值，GCC-Goto 網格測試值為 10,000、20,000、30,000，取出 3 值，以此類推，故架構 A，我們取出 240 筆數據資料，其中最佳值取出總數為 20 筆，下表以 A 架構作為說明：

A 架構		#10000				#20000				#30000				數據筆數	分區分塊最佳值 (依網格測試取出)
Tool	Blas	88	108	128	168	88	108	128	168	88	108	128	168		
HPL	Goto	3	3	3	3	-	-	-	-	-	-	-	-	12	1
	Open	3	3	3	3	-	-	-	-	-	-	-	-	12	1
GCC	Goto	3	3	3	3	3	3	3	3	3	3	3	3	36	3
	Open	3	3	3	3	3	3	3	3	3	3	3	3	36	3
	MKL	3	3	3	3	3	3	3	3	3	3	3	3	36	3
ICC	Goto	3	3	3	3	3	3	3	3	3	3	3	3	36	3
	Open	3	3	3	3	3	3	3	3	3	3	3	3	36	3
	MKL	3	3	3	3	3	3	3	3	3	3	3	3	36	3
總 數													240	20	

FIGURE 4.15: 範例：A 架構分區分塊因子取出數

於調整分區分塊因子精細度預設 88，108，128，168 下，全部架構的 80 筆最佳值分析後，發現其效能落點：E3、i7 二主機於 168 的矩陣大小的處理速度是最好的，落點統計分析如下所示，如需更進一步了解，可參考附錄 F 測試數據統計表中的「分區分塊因子統計表」。

TABLE 4.6: E3 分區分塊因子落點

Nb	Number
88	4
108	11
128	32
168	33
總數	80

TABLE 4.7: i7 分區分塊因子落點

Nb	Number
88	13
108	22
128	10
168	35
總數	80

TABLE 4.8: i7-KVM 分區分塊因子落點

Nb	Number
88	0
108	7
128	3
168	10
總數	20

#### 4.4.8 各架構效能表現

除上述各項分析，我們也可以由各架構之效能來看，當一資訊管理人員在有限的軟體資源下，可視其情況來配置其可能的架構，本論文亦提供不同架構之效能參考。

下一小節將依架構作說明與建議：

## 4.4.8.1 A 架構

A 架構為巢狀虛擬環境，效能為四架構當中最弱；因其混合使用 Workstation 與 vSphere，建議可作為教育訓練使用，受訓人員可習得二種虛擬化平台的基礎結構與安裝設置概念。

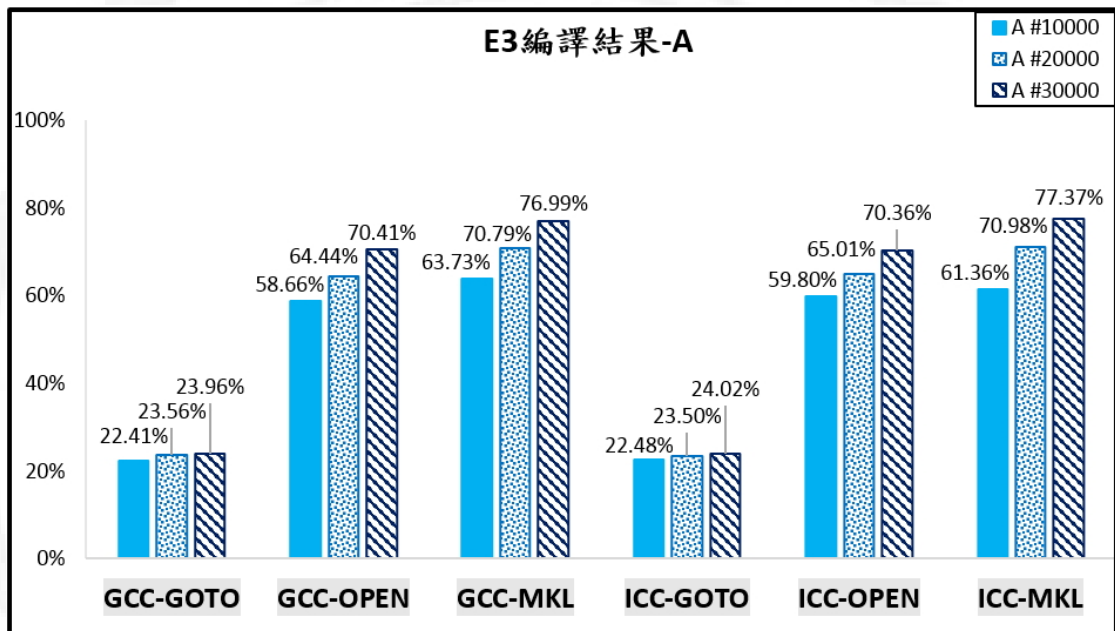


FIGURE 4.16: A 架構效能表現圖-E3

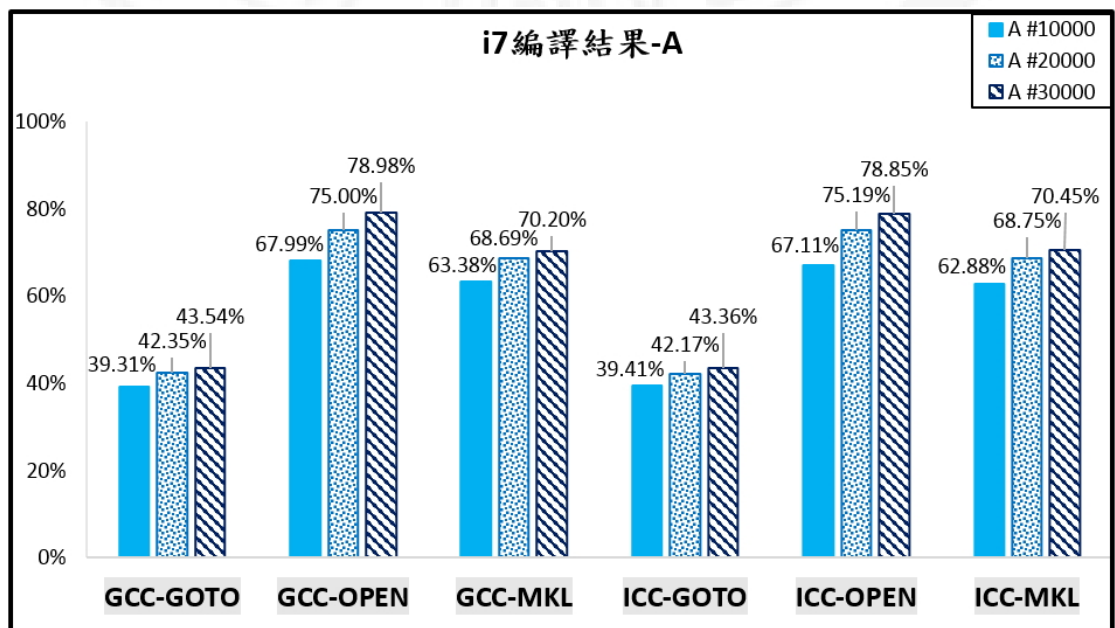


FIGURE 4.17: A 架構效能表現圖-i7

## 4.4.8.2 B 架構

B 架構為 Workstation12 環境，效能中等；建議用於一般中小型企業，在其資源及成本有限的情況下進行部署。

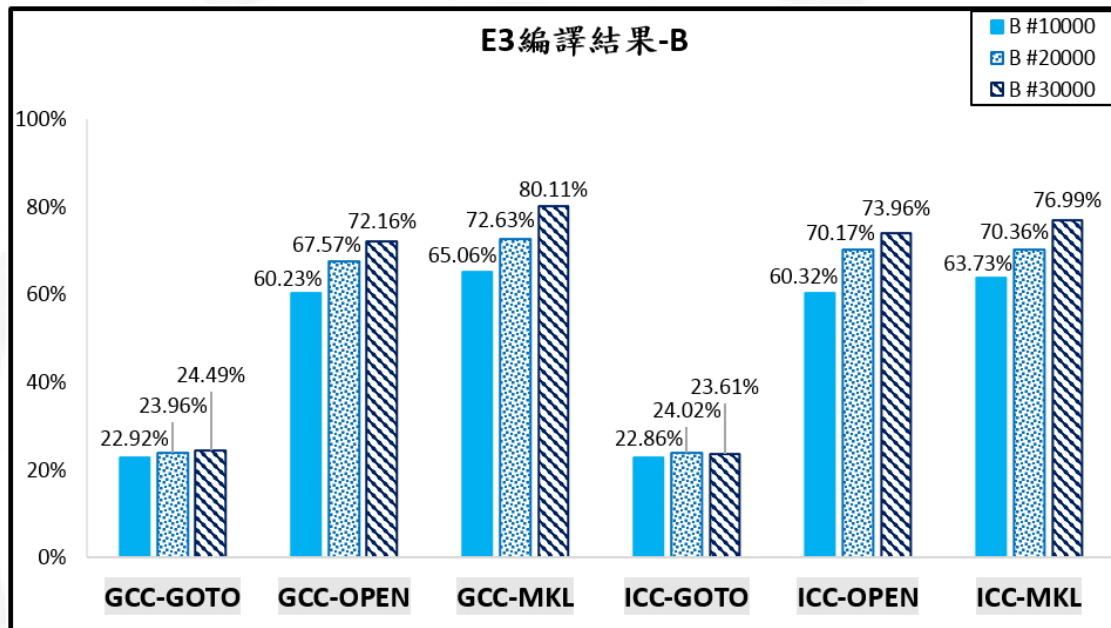


FIGURE 4.18: B 架構效能表現圖-E3

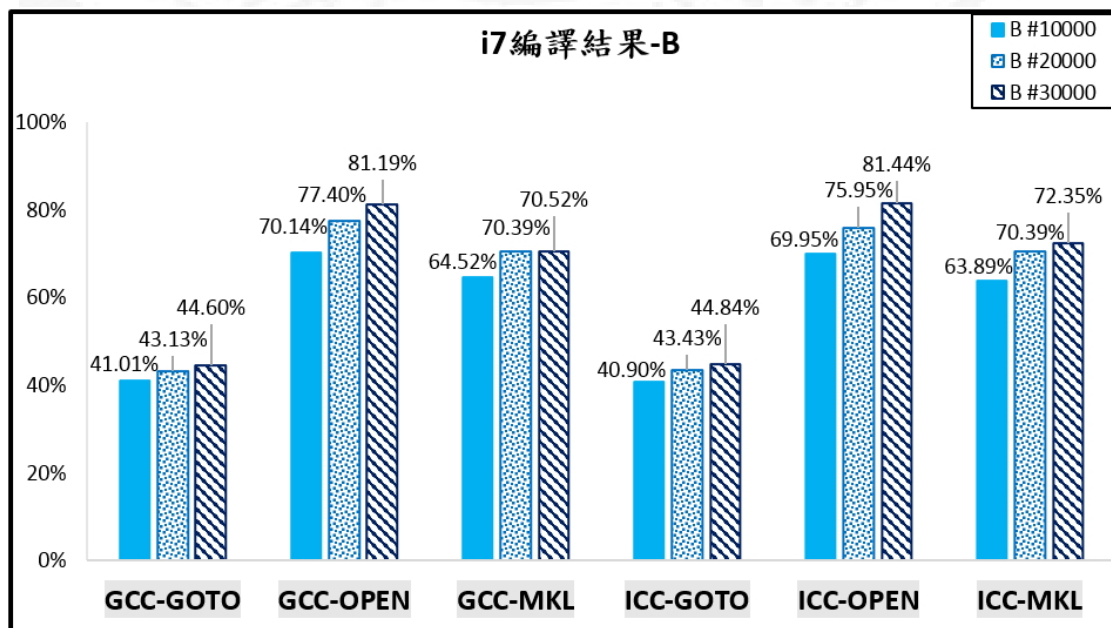


FIGURE 4.19: B 架構效能表現圖-i7



## 4.4.8.3 C 架構

C 架構為 ESXi 6 環境，效能極佳，其管理與維護的技術層面偏難，故 IT 管理者需具備資訊相關之專業知識。

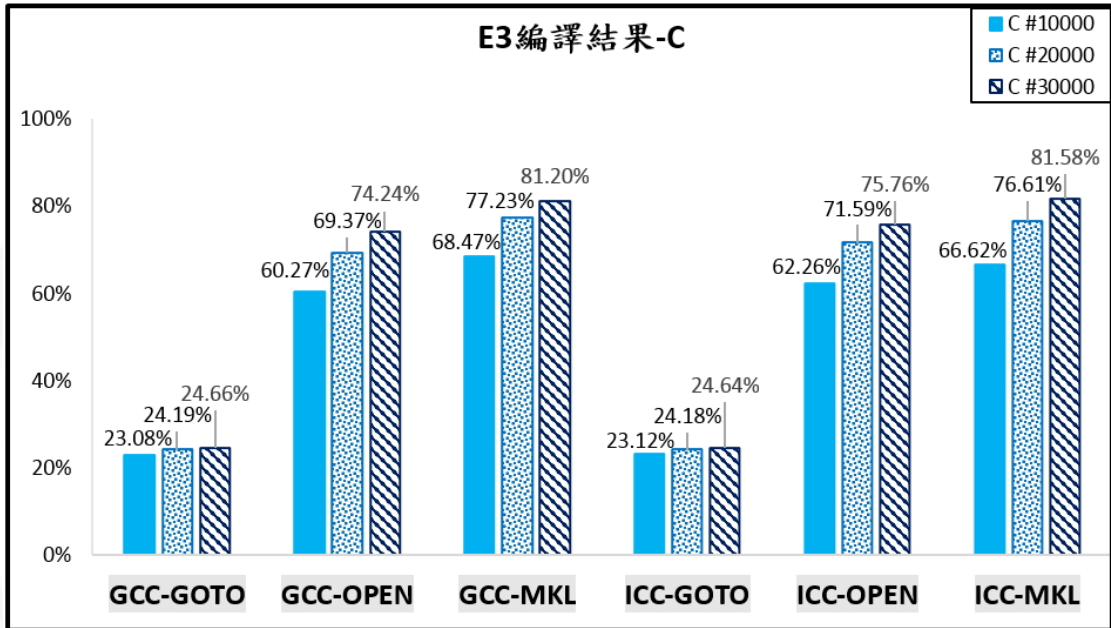


FIGURE 4.20: C 架構效能表現圖-E3

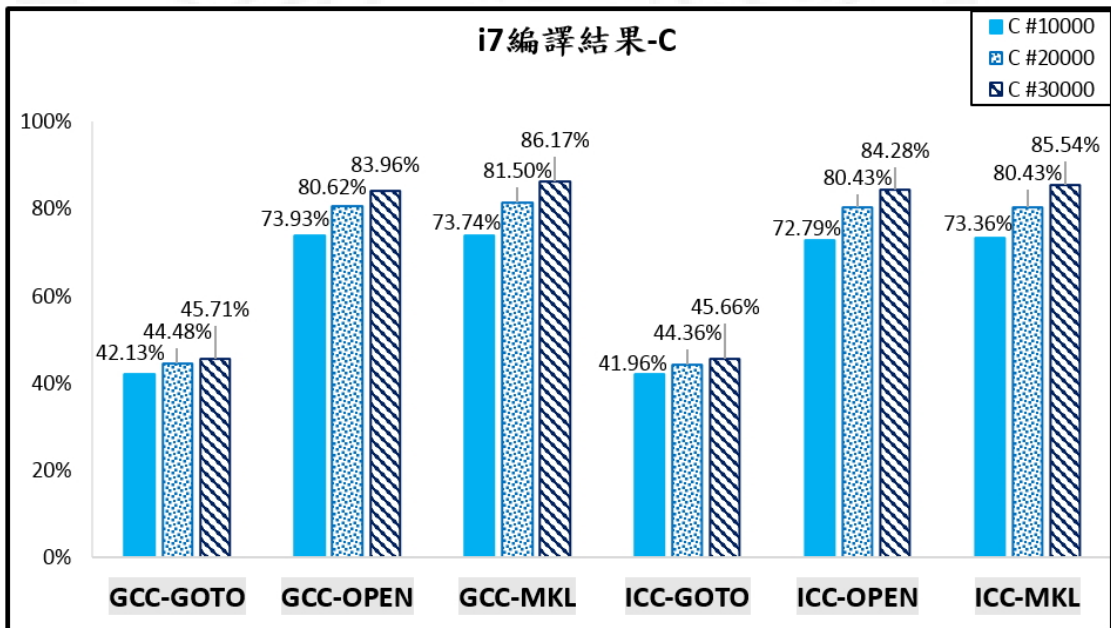


FIGURE 4.21: C 架構效能表現圖-i7

## 4.4.8.4 D 架構

D 架構為無使用虛擬化平台，效能雖佳，但單一主機無法將資源進行共享與提昇該主機的資源利用率。

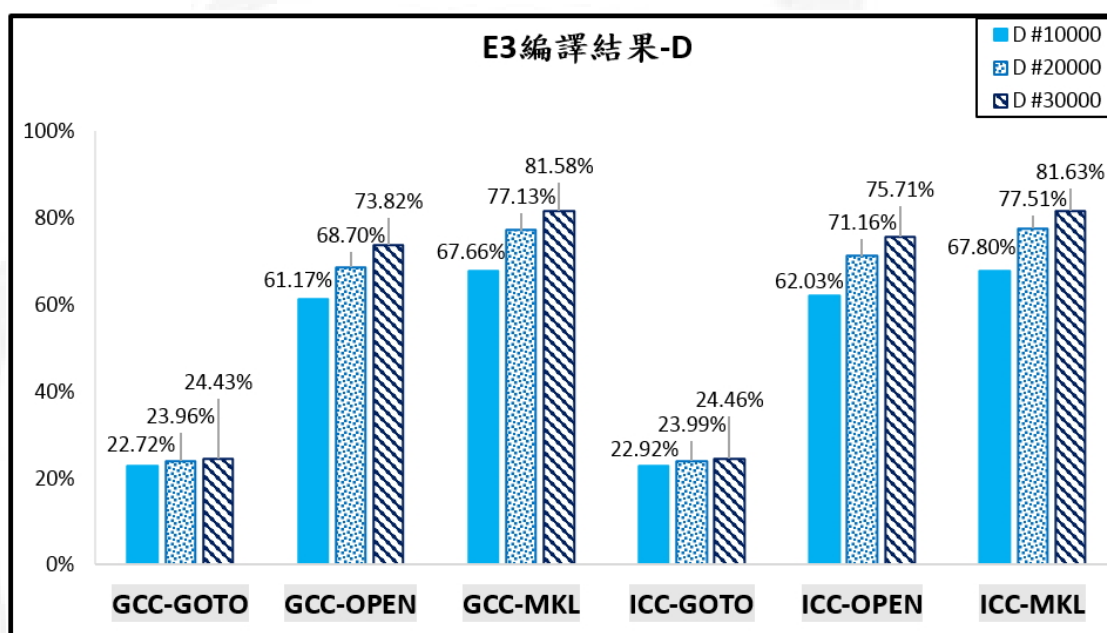


FIGURE 4.22: D 架構效能表現圖-E3

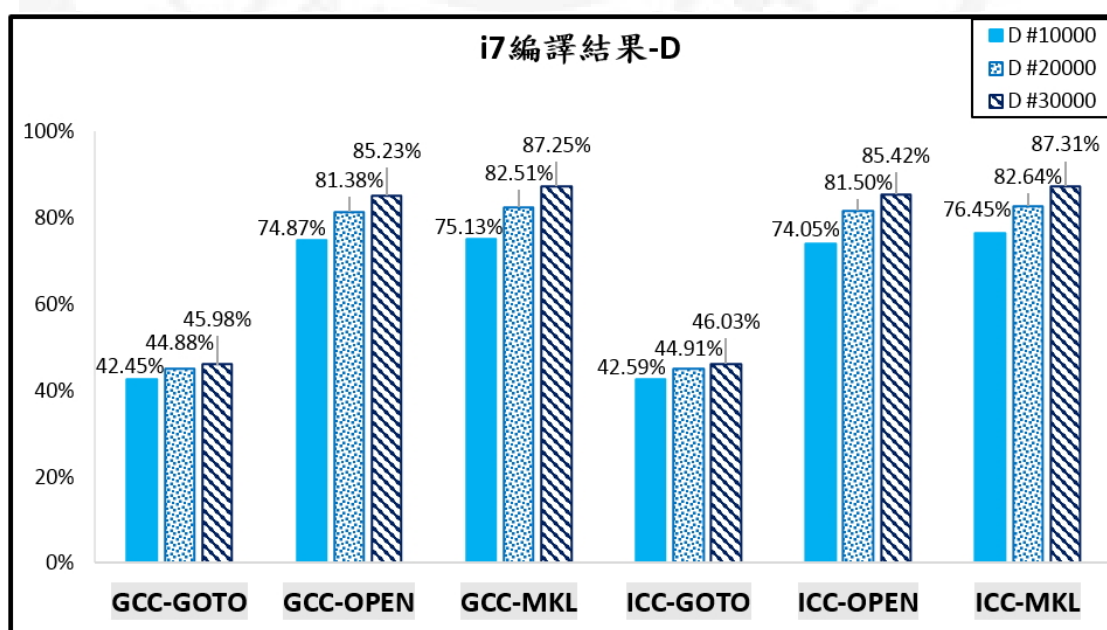


FIGURE 4.23: D 架構效能表現圖-i7



## 4.4.8.5 E 架構

E 架構為 Linux 開源虛擬化平台 KVM 環境，其優勢為效能較 A.B 架構來的好，並且能為企業節省部署虛擬化平台之費用，較可惜的是，僅能於 Linux 作業系統上安裝執行。

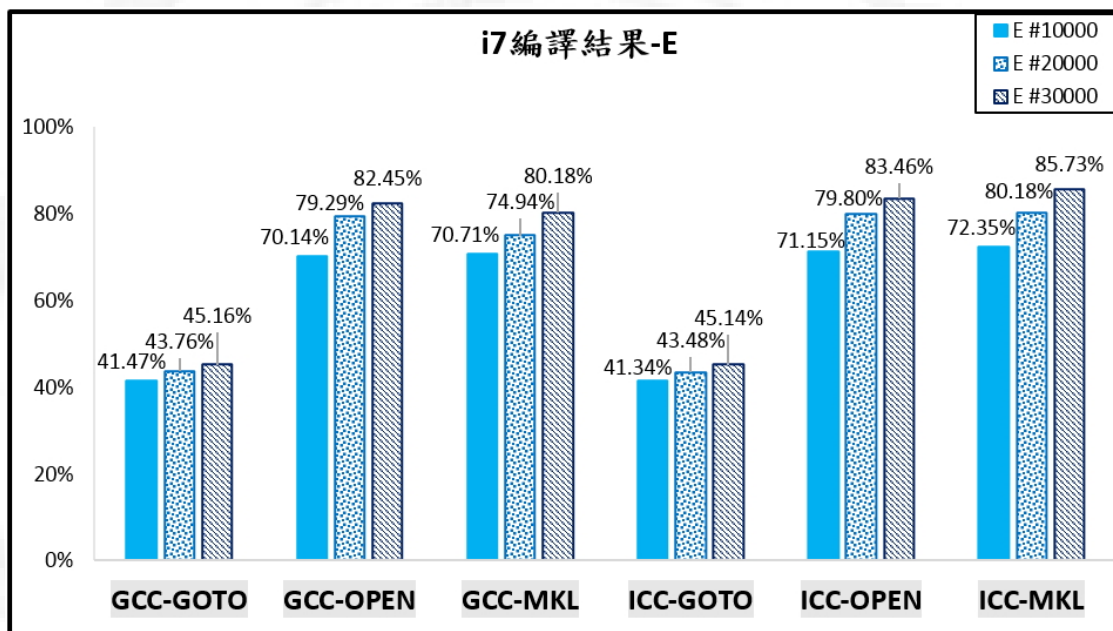


FIGURE 4.24: E 架構效能表現圖-i7

# Chapter 5

## 結論與未來方向

### 5.1 結論

最後得到下列九項結論：

- 一、由 E3-1230 V3 及 i7-3960X 主機之效能呈現，可看出 i7-3960X 效能優於 E3-1230 V3。
- 二、使用 GCC 或 ICC 編譯器對效能的影響不大。
- 三、由函式庫的角度來看，以 MKL Blas 結果最佳，其次為 Open Blas，最後為 Goto Blas。
- 四、於分區分塊因子中預設 88，108，128，168 中以 168 效能為佳，此部份設定值通常為 256 以下，並需以實測方能得到最佳值。
- 五、於網格點預設值來看 30,000 為佳，但需留意的是預設值愈高，則執行時間愈長。
- 六、由架構的部份來看，於 ESXi 架構下的表現甚佳，甚至與單純的 Linux 作業系統層（架構 D）效能相近，這意謂著 ESXi 本身耗用的資源較少，故對效能的影響不大。
- 七、當 C.D 架構優於 A.B 架構，其中之差異在於增加之作業系統層與其巢狀虛擬化之階層多寡，造成影響虛擬機效能表現的主因。
- 八、使用 Intel Linpack 測試工具來進行驗證，皆指向 C.D 架構效能表現為優。

九、如基於成本考量，KVM 架構為完全開放源所組成，其效能亦優於 A.B 架構，故 KVM 可作為企業 IT 人員於建立虛擬機的第二種選擇。

綜合上述結果，我們可以將 C 架構可作為最優配置參考依據。

## 5.2 未來方向

本論文運用主機型架構與裸機架構，在主機架構中又混合使用不同的虛擬機器管理平台 VMware Workstation 與 vSphere、KVM，裸機架構則以 ESXi 作為虛擬機管理平台；除了上述架構，另外再部署單純的主機型架構，於不使用任何虛擬機器管理平台下，實測其效能，以作為評估基準。

為強化本論文的研究目的，未來我們將考慮下列幾點來更深入的研究與探討：

- 一、編譯函式庫的部份，未來可納入其它 C 標準函式庫作為測試項目。
- 二、評估不同虛擬化管理平台，除本文中所使用的 Workstation、vSphere / ESXi、KVM 外，尚有 Docker 與 VirtualBox...等等。
- 三、雲端共享資源的部份，可嘗試如 Openstack 進行實驗，此部份除了需考量網路頻寬與安裝套件的部署。
- 四、除了實測不同架構之編譯器效能，其花費的時間也需要納入考量，當花費的時間愈長，所耗費的能源則愈多，在時間與效能之間，我們該考量到如何取得平衡點。

## 參考文獻

- [1] 林文彬. 虛擬化成本大剖析, <http://www.ithome.com.tw/node/50025>, 2008.
- [2] VMware. VMware introduction, <http://www.vmware.com/tw>, 2016.
- [3] Lizhe Wang, Dan Chen, Yangyang Hu, Yan Ma, and Jian Wang. Towards enabling cyberinfrastructure as a service in clouds. *Computers and Electrical Engineering*, 39(1):3–14, 2013. Special issue on Recent Advanced Technologies and Theories for Grid and Cloud Computing and Bio-engineering.
- [4] P. V. V. Reddy and L. Rajamani. Evaluation of different operating systems performance in the private cloud with esxi hypervisor using sigar framework. In *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference-*, pages 18–23, Sept 2014.
- [5] J. P. Walters, A. J. Younge, D. I. Kang, K. T. Yao, M. Kang, S. P. Crago, and G. C. Fox. Gpu passthrough performance: A comparison of kvm, xen, vmware esxi, and lxc for cuda and opencl applications. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 636–643, June 2014.
- [6] P. Muditha Perera and C. Keppitiyagama. A performance comparison of hypervisors. In *Advances in ICT for Emerging Regions (ICTer), 2011 International Conference on*, pages 120–120, Sept 2011.
- [7] R. Bakhshayeshi, M. K. Akbari, and M. S. Javan. Performance analysis of virtualized environments using hpc challenge benchmark suite and analytic

- hierarchy process. In *Intelligent Systems (ICIS), 2014 Iranian Conference on*, pages 1–6, Feb 2014.
- [8] 中央研究院計算中心. Linpack 使用手冊, <http://www.ascc.sinica.edu.tw/pd-man/linpack/linpack.html>, 2016.
- [9] G. Jo, J. Nah, J. Lee, J. Kim, and J. Lee. Accelerating linpack with mpi-opencl on clusters of multi-gpu nodes. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):1814–1825, July 2015.
- [10] Feng Wang, Canqun Yang, and Juncheng Bai. Hybrid mpi/openmp optimization in linpack benchmark on multi-core platforms. In *Computer Science Education (ICCSE), 2013 8th International Conference on*, pages 917–920, April 2013.
- [11] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov, G. Henry, A. G. Shet, G. Chrysos, and P. Dubey. Design and implementation of the linpack benchmark for single and multi-node systems based on intel x00ae; xeon phi coprocessor. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 126–137, May 2013.
- [12] C. T. Yang, K. C. Wang, H. Y. Cheng, C. T. Kuo, and W. C. C. Chu. Green power management with dynamic resource allocation for cloud virtual machines. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 726–733, Sept 2011.
- [13] Guodong Liu and Xiangdong Hu. Performance and efficiency evaluation and analysis of supercomputers. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 4, pages 642–646, July 2010.
- [14] Arquimedes Canedo, Ben A. Abderazek, and Masahiro Sowa. Design and implementation of a queue compiler. *Microprocessors and Microsystems*, 33(2): 129–138, 2009.

- [15] Yu-Lin Sung, Chao-Tung Yang, Shuo-Tsung Chen, and Kuang-Chin Chang. Evaluating virtual cluster configuration and compiler measured. In *Trustworthy Systems and Their Applications (TSA), 2015 Second International Conference on*, pages 89–93, July 2015.
- [16] D. Bokan, M. Đukić, M. Popović, and N. Četić. Adjustment of gcc compiler frontend for embedded processors. In *Telecommunications Forum Telfor (TELFOR), 2014 22nd*, pages 983–986, Nov 2014.
- [17] Joseph Yiu. Chapter 20 - getting started with the {GNU} c compiler. In Joseph Yiu, editor, *The Definitive Guide to the {ARM} Cortex-M0*, pages 385–403. Newnes, Oxford, 2011.
- [18] Dmitry Plotnikov, Dmitry Melnik, Mamikon Vardanyan, Ruben Buchatskiy, Roman Zhuykov, and Je-Hyung Lee. Automatic tuning of compiler optimizations and analysis of their impact. *Procedia Computer Science*, 18:1312–1321, 2013. 2013 International Conference on Computational Science.
- [19] Intel. Intel parallel studio xe, <https://software.intel.com/en-us/intel-parallel-studio-xe>, 2016.
- [20] Manuel F. Dolz, Francisco D. Igual, Thomas Ludwig, Luis Piñuel, and Enrique S. Quintana-Ortí. Balancing task- and data-level parallelism to improve performance and energy consumption of matrix computations on the intel xeon phi. *Computers and Electrical Engineering*, 46:95–111, 2015.
- [21] S. P. Levitt. C++: an evolving language. In *AFRICON, 2004. 7th AFRICON Conference in Africa*, volume 2, pages 1197–1202 Vol.2, Sept 2004.
- [22] S. Pellegrini, R. Prodan, and T. Fahringer. A lightweight c++ interface to mpi. In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, pages 3–10, Feb 2012.

- [23] H. Winroth. A scripting language interface to c++ libraries. In *Technology of Object-Oriented Languages and Systems, 1997. TOOLS 23. Proceedings*, pages 247–259, Jul 1997.
- [24] H. Carr, R. R. Kessler, and M. Swanson. Compiling distributed c++. In *Parallel and Distributed Processing, 1993. Proceedings of the Fifth IEEE Symposium on*, pages 496–503, Dec 1993.
- [25] N. Tripathi and D. B. Ojha. An application to help unskilled user to learn c++ a programming language learning tool. In *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, volume 3, pages V3–457–V3–460, June 2010.
- [26] P. Schweitzer, C. Mazel, D. R. C. Hill, and C. Cârloganu. Inputs of aspect oriented programming for the profiling of c++ parallel applications on many-core platforms. In *High Performance Computing Simulation (HPCS), 2014 International Conference on*, pages 793–802, July 2014.
- [27] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Comput.*, 22(6):789–828, sep 1996.
- [28] CORPORATE The MPI Forum. Mpi: A message passing interface. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, Supercomputing.93, pages 878–883, New York, NY, USA, 1993. ACM.
- [29] Ashwin M. Aji, Lokendra S. Panwar, Feng Ji, Milind Chabbi, Karthik Murthy, Pavan Balaji, Keith R. Bisset, James Dinan, Wu-chun Feng, John Mellor-Crummey, Xiaosong Ma, and Rajeev Thakur. On the efficacy of gpu-integrated mpi for scientific applications. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, pages 191–202, New York, NY, USA, 2013. ACM.

- [30] Ron Brightwell and Keith D. Underwood. An analysis of the impact of mpi overlap and independent progress. In *Proceedings of the 18th Annual International Conference on Supercomputing, ICS '04*, pages 298–305, New York, NY, USA, 2004. ACM.
- [31] A. Elsayed and N. Abdelbaki. Performance evaluation and comparison of the top market virtualization hypervisors. In *Computer Engineering Systems (ICCES) , 2013 8th International Conference on*, pages 45–50, Nov 2013.
- [32] E. Angerson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof, and D. Sorensen. Lapack: A portable linear algebra library for high-performance computers. In *Supercomputing '90., Proceedings of*, pages 2–11, Nov 1990.
- [33] S. Varrette, M. Guzek, V. Plugaru, X. Besseron, and P. Bouvry. Hpc performance and energy-efficiency of xen, kvm and vmware hypervisors. In *2013 25th International Symposium on Computer Architecture and High Performance Computing*, pages 89–96, Oct 2013.
- [34] C. Mancaş. Performance improvement through virtualization. In *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER) ,* pages 253–256, Sept 2015.
- [35] R. Bakhshayeshi, M. K. Akbari, and M. S. Javan. Performance analysis of virtualized environments using hpc challenge benchmark suite and analytic hierarchy process. In *Intelligent Systems (ICIS) , 2014 Iranian Conference on*, pages 1–6, Feb 2014.
- [36] O. Z. Qiu and Z. Yue. Research on application of virtualization in network technology course. In *Computer Science Education (ICCSE) , 2012 7th International Conference on*, pages 357–359, July 2012.



- [37] Michael A. Penhallurick. Methodologies for the use of {VMware} to boot cloned/mounted subject hard disk images. *Digital Investigation*, 2(3):209 – 222, 2005.
- [38] Xiaofei Liao, Hai Jin, and Haikun Liu. Towards a green cluster through dynamic remapping of virtual machines. *Future Generation Computer Systems*, 28(2):469 – 477, 2012.
- [39] Keyvan RahimiZadeh, Morteza AnaLoui, Peyman Kabiri, and Bahman Javadi. Performance modeling and analysis of virtualized multi-tier applications under dynamic workloads. *Journal of Network and Computer Applications*, 56:166 – 187, 2015.
- [40] Hai Jin, Wei Gao, Song Wu, Xuanhua Shi, Xiaoxin Wu, and Fan Zhou. Optimizing the live migration of virtual machine by {CPU} scheduling. *Journal of Network and Computer Applications*, 34(4):1088 – 1096, 2011. Advanced Topics in Cloud Computing.
- [41] A. Anand, M. Dhingra, J. Lakshmi, and S. K. Nandy. Resource usage monitoring for kvm based virtual machines. In *Advanced Computing and Communications (ADCOM) , 2012 18th Annual International Conference on*, pages 66–70, Dec 2012.
- [42] J. Liu and Q. Hao. Research on optimizing kvm’s network performance. In *Internet Technology and Applications (iTAP) , 2011 International Conference on*, pages 1–4, Aug 2011.
- [43] S. G. Soriga and M. Barbulescu. A comparison of the performance and scalability of xen and kvm hypervisors. In *Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition*, pages 1–6, Sept 2013.
- [44] R. K T. Virtual cpu scheduling techniques for kernel based virtual machine (kvm) . In *Cloud Computing in Emerging Markets (CCEM) , 2013 IEEE International Conference on*, pages 1–6, Oct 2013.

- [45] D. Beserra, F. Oliveira, J. Araujo, F. Fernandes, A. Araújo, P. Endo, P. Maciel, and E. D. Moreno. Performance evaluation of hypervisors for hpc applications. In *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*, pages 846–851, Oct 2015.



## 附錄 A

### 安裝 Ubuntu

步驟 1. 開機時連續點擊「F8」，選擇啟動類型：USB。

步驟 2. 緊接著按下「空白鍵」。

步驟 3. 選擇「安裝 Ubuntu」。



FIGURE A.1: Ubuntu 安裝畫面

步驟 4. 進入歡迎畫面，中文 (繁體)，按下「繼續」。

步驟 5. 將「當安裝時下載更新」勾選，按下「繼續」。

步驟 6. 安裝類型「其他」，自訂分割，按下「繼續」。

- 步驟 7. 將「/dev/sda1」設為所需之硬碟分割「Ext4 日誌式檔案系統」，掛載點為「/」，按下「OK」。
- 步驟 8. 將「/dev/sda2」設為所需之虛擬記憶體「置換空間」，按下「OK」。
- 步驟 9. 點選「/dev/sda1 Ext4」作為選定之安裝目的，按下「立即安裝」。如出現警告訊息，請點選「繼續」。
- 步驟 10. 選擇時區「Taipei」。
- 步驟 11. 鍵盤排列方式「英語(美式)」，這會影響到輸入符號與按鍵預設位置，按下「繼續」。
- 步驟 12. 輸入您的使用者名稱與密碼後，按下「繼續」。
- 步驟 13. 進行安裝。
- 步驟 14. 完成安裝，點選「立即重新啟動」，此時也可以拔除 USB。
- 步驟 15. 重新開機完成，登入畫面，請輸入您的密碼。

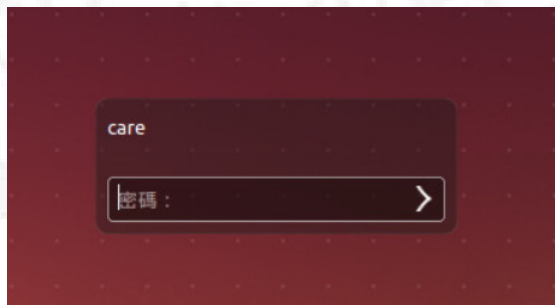


FIGURE A.2: Ubuntu 登入畫面

- 步驟 16. 搜尋您的電腦和線上來源，將 Terminal(終端機)拉到桌面工具列中。



FIGURE A.3: 終端機 Terminal

步驟 17. 至 Ubuntu 軟體中心中，依需求安裝「p7.zip」…等，  
需輸入密碼。

步驟 18. 依需求安裝「ssh、vim」…等。

指令：`sudo apt-get install -y ssh`

指令：`sudo apt-get install -y vim`

步驟 19. 更新線上套件庫資訊

指令：`sudo apt-get update`

步驟 20. 執行系統更新

指令：`sudo apt-get upgrade`

步驟 21. 重新啟動 Ubuntu。

## A.1 Ubuntu 下安裝 KVM

步驟 1. 開啟 terminal。

步驟 2. 查詢是否支援虛擬化技術。

指令：`egrep -c '(sum|vmx)' /proc/cpuinfo`

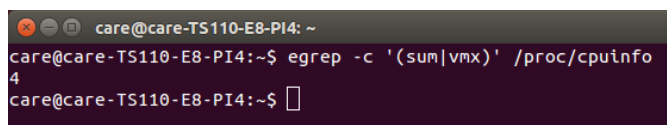


FIGURE A.4: 查詢有無支援虛擬化技術

步驟 3. 安裝 KVM。

指令：sudo apt-get install -y qemu-kvm

libvirt-bin bridge-utils virt-manager

鍵入：password

鍵入：y

```
care@care-TS110-E8-PI4:~$ sudo apt-get install -y qemu-kvm libvirt-bin bridge-utils virt-manager
[sudo] password for care:
正在讀取套件清單... 完成
正在重建相依關係... 完成
正在讀取狀態資料... 完成
下列的額外套件將被安裝：
augeas-lenses cgroup-lite cpu-checker ebttables gawk ipxe-qemu libaio1
libappindicator1 libaugeas0 libbonoboui2-0 libbonoboui2-common
libboost-thread1.54.0 libfdt1 libglade2-0 libgnomecanvas2-0
libgnomecanvas2-common libgnomeui-0 libgnomeui-common libgtk-vnc-1.0-0
libgvnc-1.0-0 libindicator7 libmetcrf1 librados2 librbdi libsd11.zdeb1an
libseccomp2 libsgsgv2 libspice-server1 libusbredirparser1 libvirt0
libvte-common libvte9 libxen-4.4 libxenstore3.0 libxml2-utils msr-tools
python-appindicator python-glade2 python-gnome2 python-gtk-vnc
python-libvirt python-pycurl python-pyorbitt python-urllibgrabber python-vte
qemu-keymaps qemu-system-common qemu-system-x86 qemu-utils seabios sharutils
virtinst
建議套件：
augeas-doc gawk-doc augeas-tools radvd lvm2 python-gtk2-doc
python-gnome2-doc libcurl4-gnutls-dev python-pycurl-dbg python-pyorbitt-dbg
samba vde2 sgablos kvm-lpxe-precise debootstrap bsd-mailx mailx virt-viewer
python-guestfs python-spice-client-gtk
下列【新】套件將被安裝：
augeas-lenses bridge-utils cgroup-lite cpu-checker ebttables gawk ipxe-qemu
libaio1 libappindicator1 libaugeas0 libbonoboui2-0 libbonoboui2-common
libboost-thread1.54.0 libfdt1 libglade2-0 libgnomecanvas2-0
libgnomecanvas2-common libgnomeui-0 libgnomeui-common libgtk-vnc-1.0-0
libgvnc-1.0-0 libindicator7 libmetcrf1 librados2 librbdi libsd11.zdeb1an
```

FIGURE A.5: 安裝 KVM

步驟 4. 確認 KVM 執行狀態。

指令：sudo virsh -c qemu:///system list

指令：lsmod | grep kvm

```
care@care-TS110-E8-PI4:~$ sudo virsh -c qemu:///system list
Id          名稱          狀態
-----
care@care-TS110-E8-PI4:~$
```

FIGURE A.6: 確認 KVM 執行狀態 1

```
care@care-TS110-E8-PI4:~$ lsmod |grep kvm
kvm_intel    151552  0
kvm          479232  1 kvm_intel
care@care-TS110-E8-PI4:~$
```

FIGURE A.7: 確認 KVM 執行狀態 2

步驟 5. 開啟 KVM 管理程序。



FIGURE A.8: KVM 管理程序 1

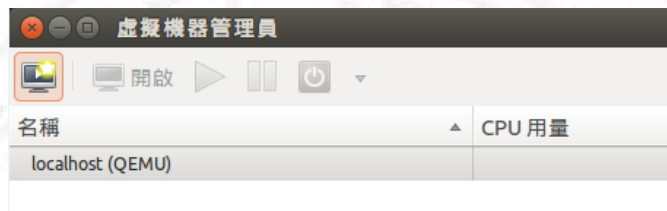


FIGURE A.9: KVM 管理程序 2

步驟 6. 建立虛擬機 ubuntu。



FIGURE A.10: 建立虛擬機 1



FIGURE A.11: 建立虛擬機 2



FIGURE A.12: 建立虛擬機 3





FIGURE A.13: 建立虛擬機 4



FIGURE A.14: 建立虛擬機 5

## 附錄 B

# 安裝 VMware vSphere

### B.1 安裝 VMware vSphere

步驟 1. 開機時連續點擊「F8」選擇啟動類型：USB。

步驟 2. 進行安裝「ESXi」。



FIGURE B.1: ESXi 安裝畫面

步驟 3. 選擇要執行的操作，按下「Enter」鍵。

步驟 4. 使用者協議，按下「F11」鍵。

步驟 5. 選擇安裝磁碟或是升級：

步驟 6. 確認磁盤選擇，按下「Enter」鍵。

確認已選擇至少包含一個分區與現有數據的磁盤，所選的磁盤  
將被覆蓋。

步驟 7. 選擇鍵盤佈局。

選擇「US Default」，按下「Enter」鍵。

步驟 8. 鍵入管理員密碼後，按下「Enter」鍵。

註：至少中英文共 7 碼。

步驟 9. 確認安裝，按下「F11」鍵。

將出現警告：該磁盤將被重新分區。

步驟 10. 安裝完成，按下「Enter」鍵，自動重啟服務器。

步驟 11. 按下「F2」鍵，進行身份驗證。

登入名稱：root

密碼：同步驟 8

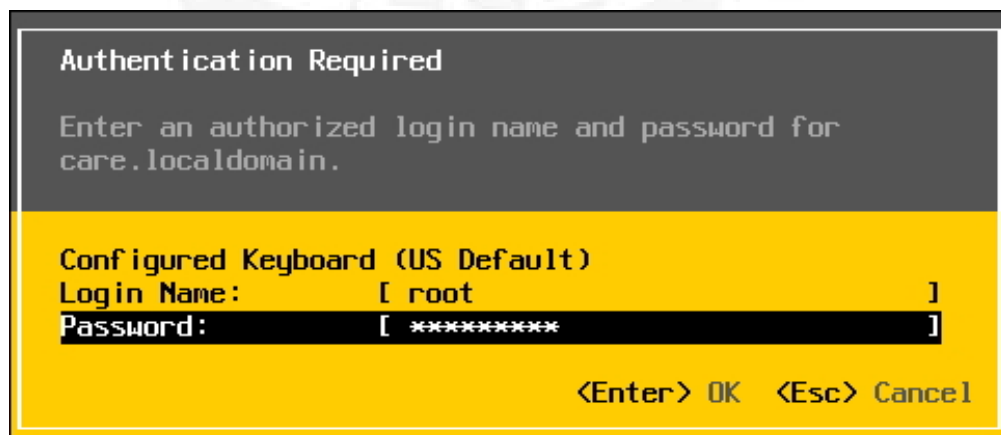


FIGURE B.2: ESXi 身份驗證畫面

步驟 12. 網路設定 (IPv4)。

選擇 Configure Management Network

Use dynamic Ipv4 address and network configuration.

確定後，可先 [Esc]，再重新進入此選項就會代入 IP，

改選 Set static Ipv4 address and network configuration.

步驟 13 網路設定 (Set staticIPv6)。

設定 IPv6：Disable IPv6 (restart required)。

步驟 14. 網路設定 (DNS)

設定 DNS Configuration：

1)Primary DNS Server：自訂。

2)Hostname：自訂。

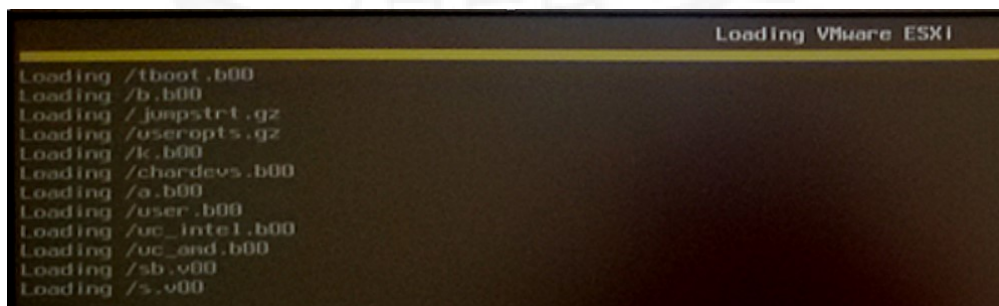
步驟 15. 按下「Esc」鍵儲存設定，按下「y」離開並重啟網路設定。

步驟 16. 網路設定 (SSH)。

選擇 Troubleshooting Options。

選擇『Enabled / Disable SSH』，然後按鍵盤『Enter』來切換  
啟用或是關閉。(建議啟用 SSH)

步驟 17. 載入各項設定值。



```
Loading VMware ESXi
Loading /tboot.b00
Loading /b.b00
Loading /jumpstrt.gz
Loading /useropts.gz
Loading /k.b00
Loading /chardevs.b00
Loading /a.b00
Loading /user.b00
Loading /uc_intel.b00
Loading /uc_and.b00
Loading /sb.v00
Loading /s.v00
```

FIGURE B.3: ESXi 載入設定畫面

步驟 18. 使用另一電腦進行連線測試，開啟瀏覽器，輸入下載網址：

<https://my.vmware.com/web/vmware/info/slug/>

選擇下載 VMware vCenter Server。

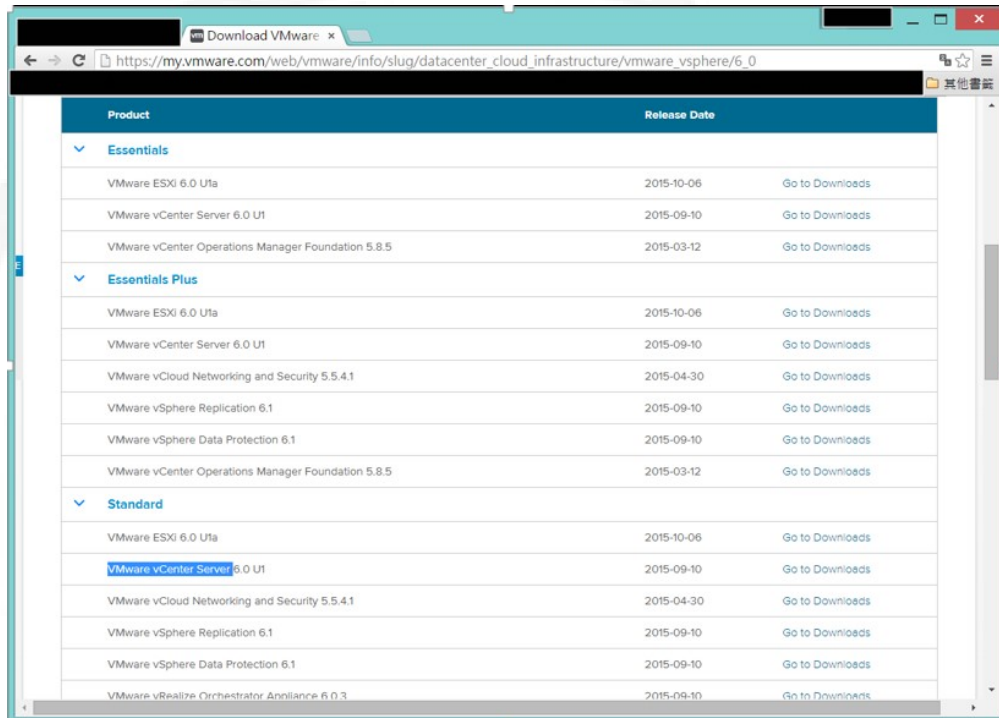


FIGURE B.4: ESXi 下載網址畫面

步驟 19. 下載完成後，解壓縮並執行「VMware-viclient-all-6.0.0.exe」

步驟 20. 選擇語言：自訂。

步驟 21. 安裝 VMware vSphere Client 6.0，點選下一步。



FIGURE B.5: vSphere Client 安裝畫面

步驟 22. 使用者授權：點選「接受」，進行下一步。

步驟 23. 選擇目的地資料夾。

步驟 24. 安裝完成後，開啟應用程式「VMware vSphere Client」。

1) 輸入 IP 位址：主機端 IP 位址

2) 使用者名稱：root

密碼：自訂

安全性警告「略過」

步驟 25. 授權認證：點選「確定」。

步驟 26. 進入 VMware vSphere Client。

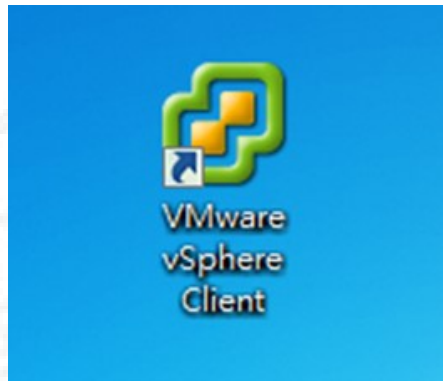


FIGURE B.6: vSphere Client 桌面 icon

步驟 27. 輸入授權碼。

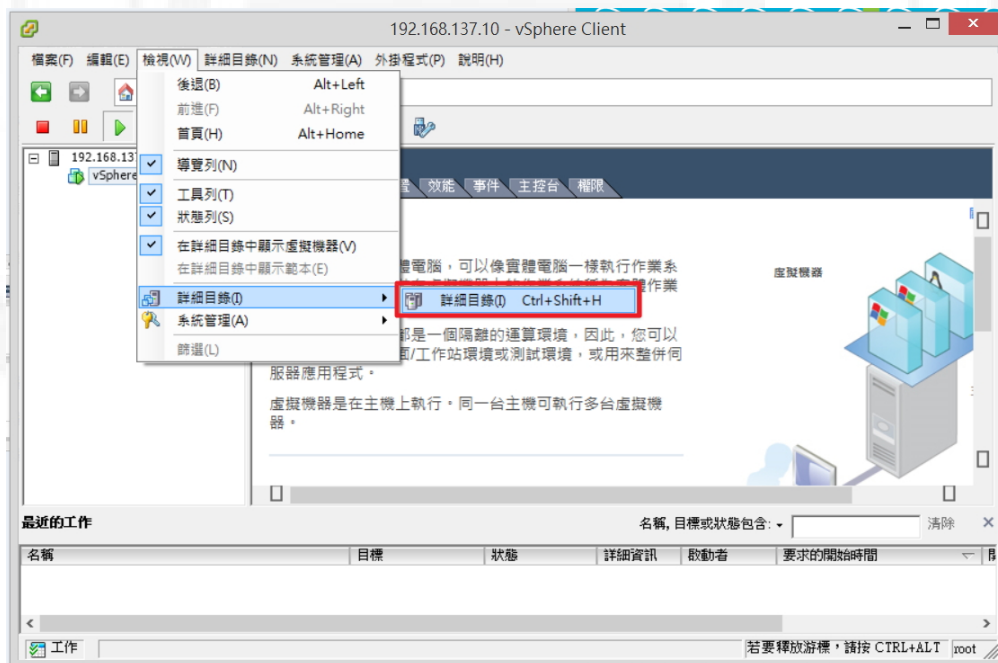


FIGURE B.7: vSphere Client 驗證授權碼畫面 1



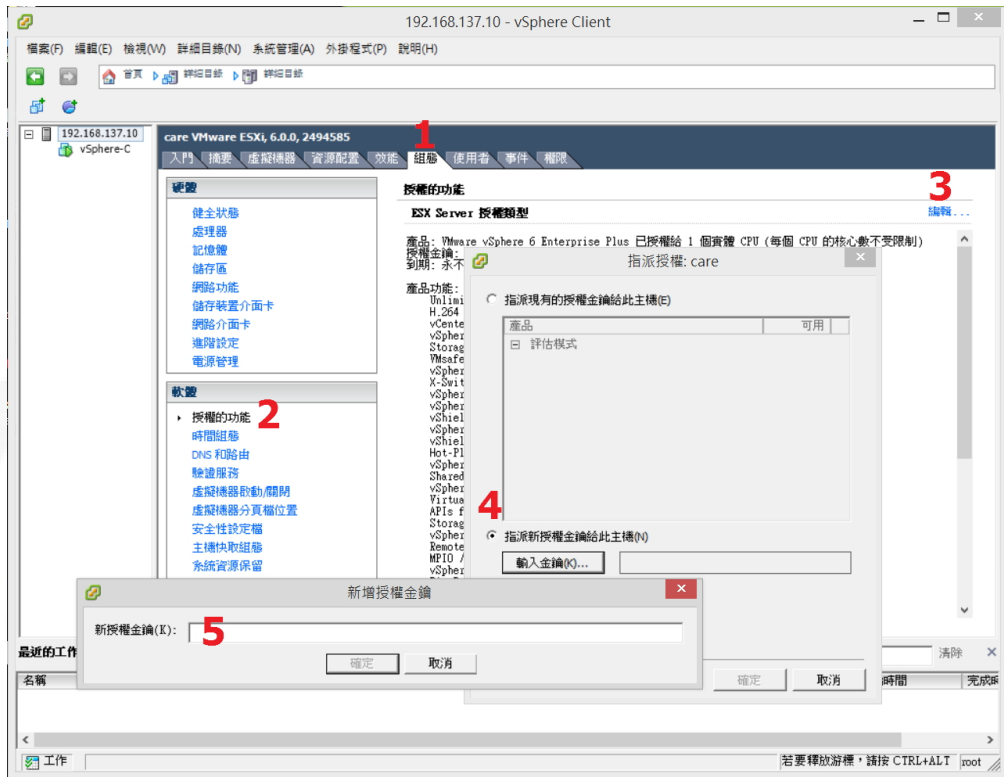


FIGURE B.8: vSphere Client 驗證授權碼畫面 2

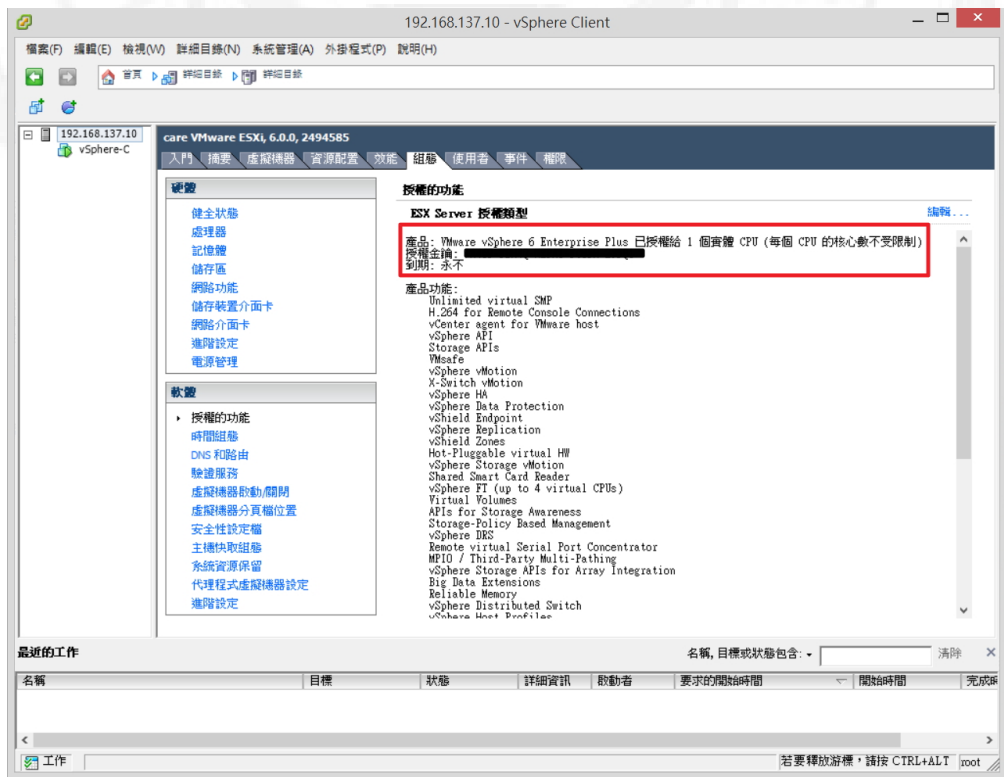


FIGURE B.9: vSphere Client 驗證授權碼畫面 3

步驟 28. 如需結束 ESXi :

- 1) 按下「F12」鍵
- 2) 輸入帳號/密碼
- 3) 按下「F2」鍵

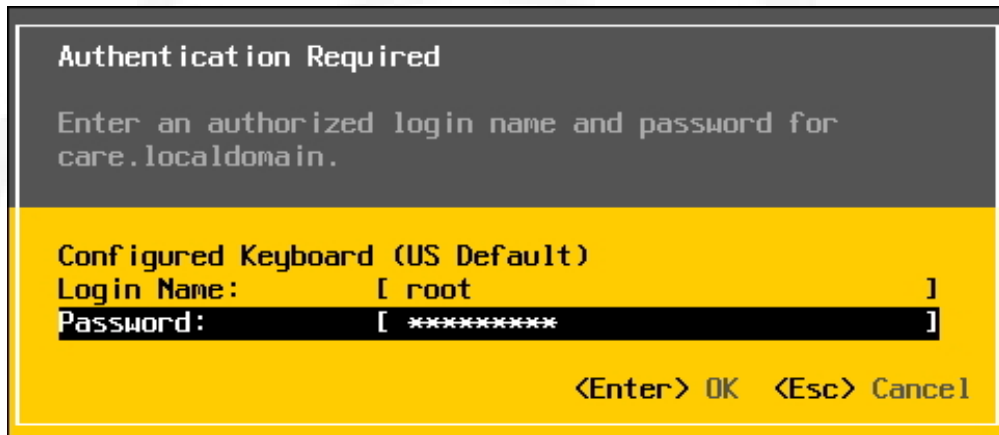


FIGURE B.10: 結束 ESXi 畫面

## B.2 於 VMware vSphere 6 下安裝 Ubuntu

步驟 1. 設定 NTP 伺服器，使時間同步。

新增：clock.stdtime.gov.tw

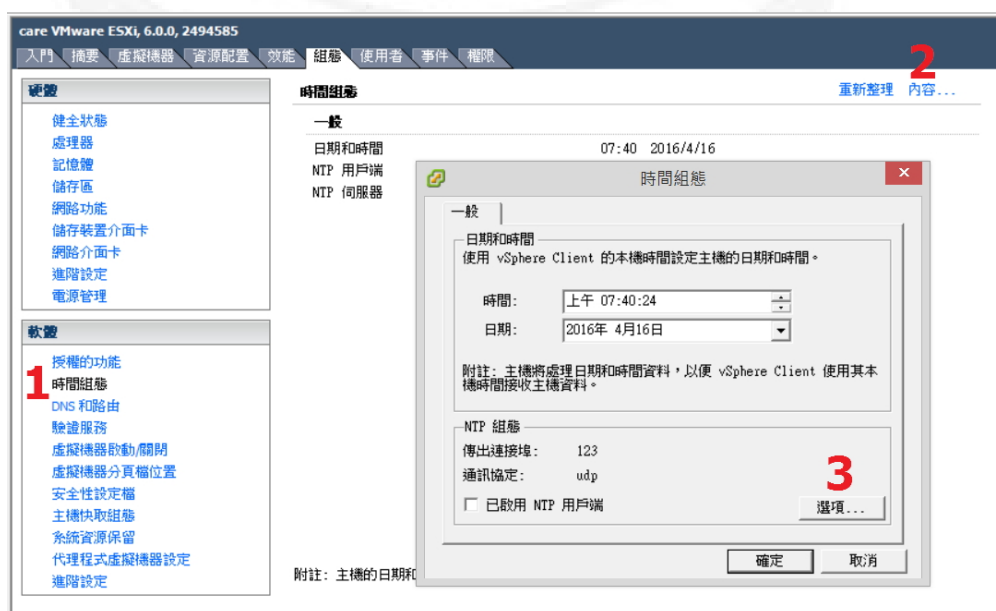


FIGURE B.11: 新增：clock.stdtime.gov.tw 1

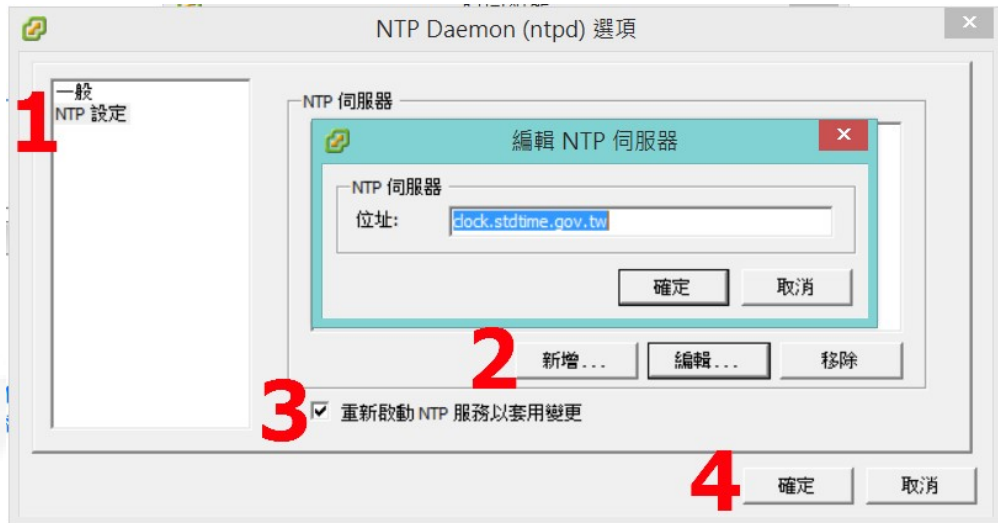


FIGURE B.12: 新增：clock.stdtime.gov.tw 2



FIGURE B.13: 新增：clock.stdtime.gov.tw 3

步驟 2. 上傳必需之 iso 檔案。

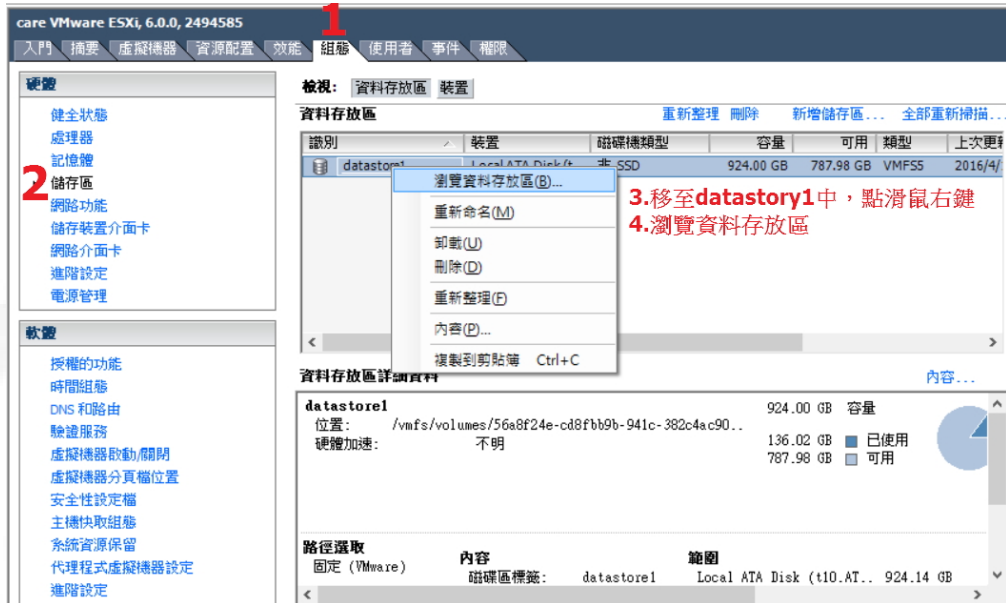


FIGURE B.14: 新增：上傳 iso 檔案 1

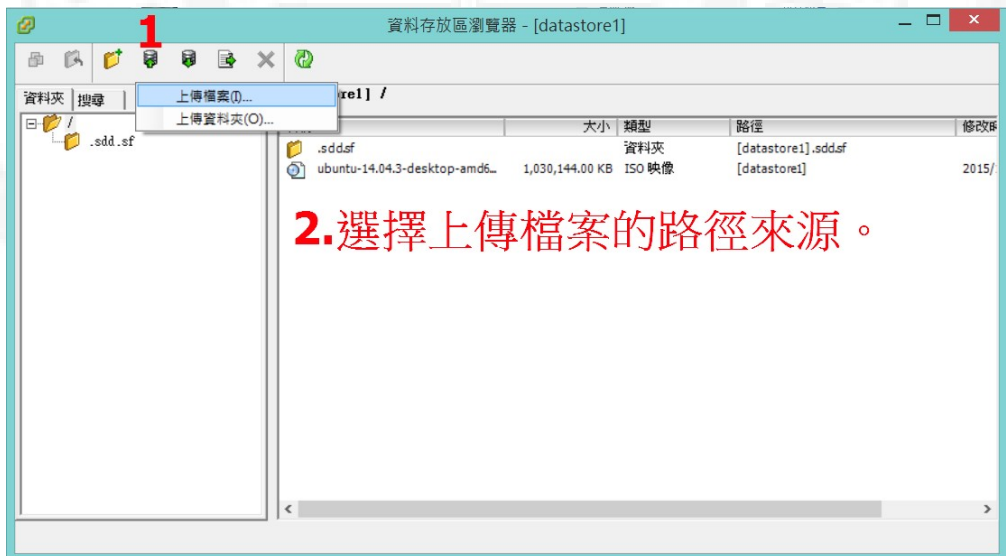


FIGURE B.15: 新增：上傳 iso 檔案 2

### 步驟 3. 建立新的虛擬機器。



FIGURE B.16: 建立新的虛擬機器 1

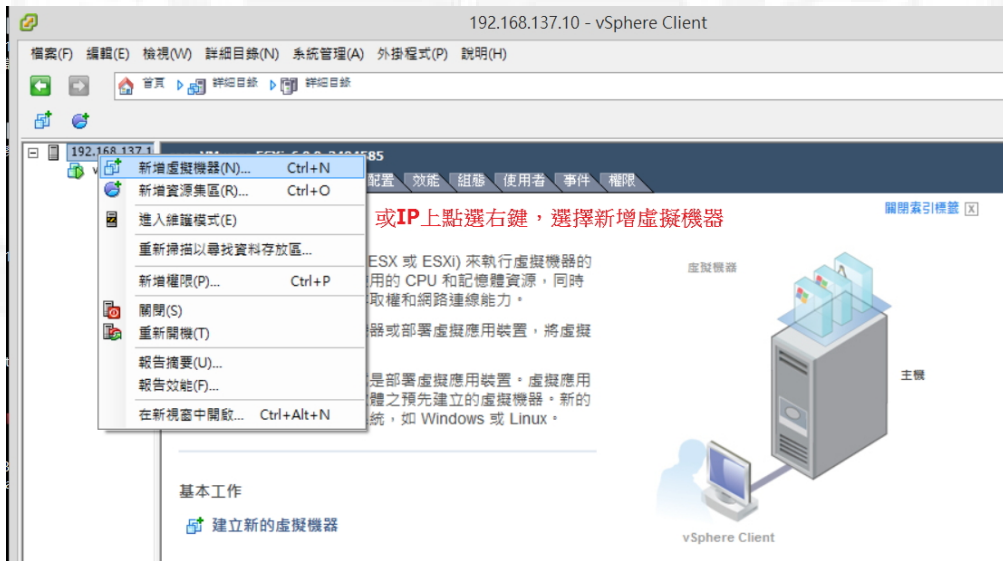


FIGURE B.17: 建立新的虛擬機器 2

步驟 4. 選取虛擬機器的組態，點選「自訂」進行下一步。

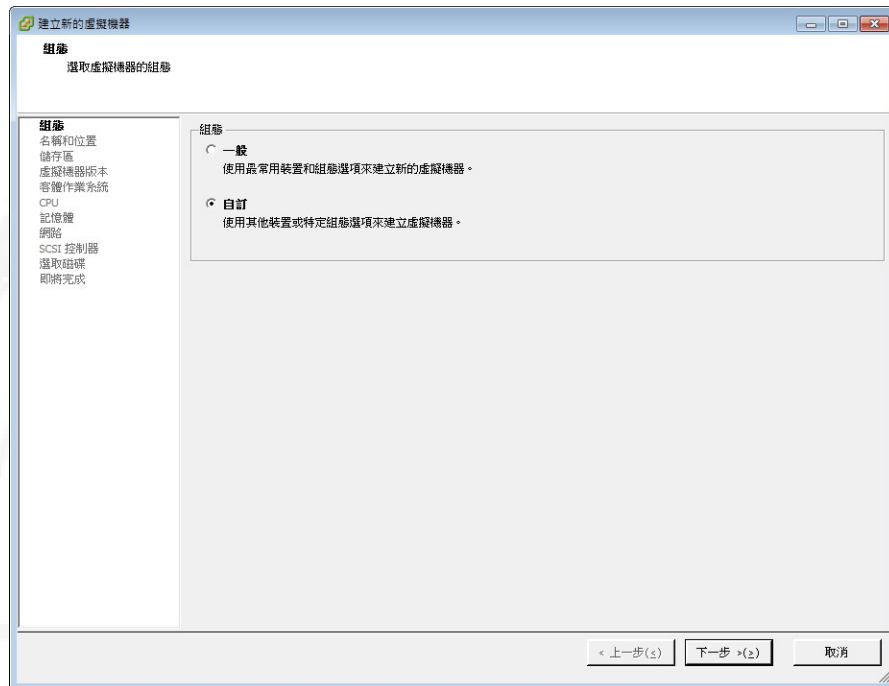


FIGURE B.18: 虛擬機器組態

步驟 5. 指定虛擬機器的名稱和位置。

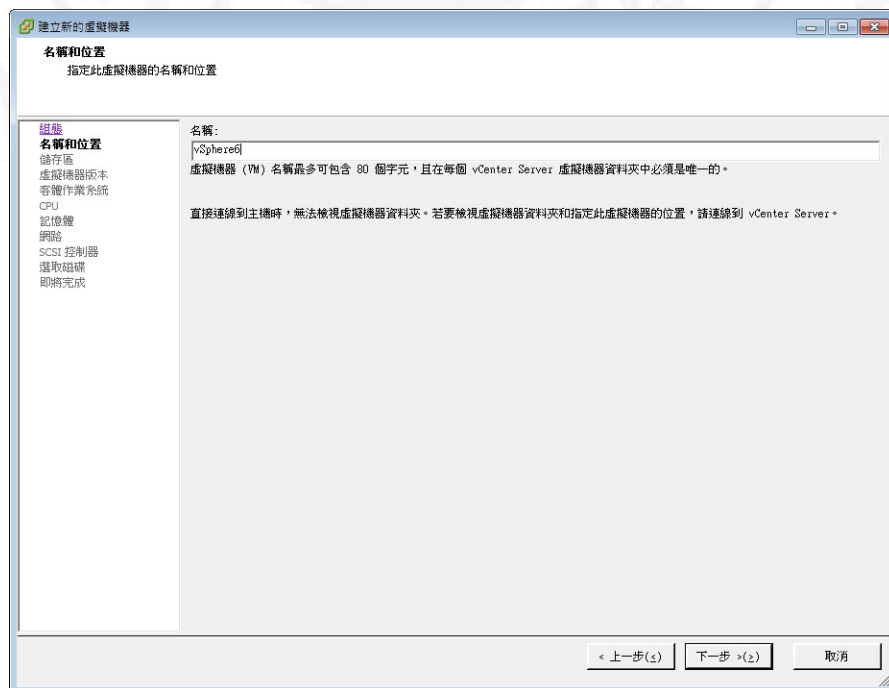


FIGURE B.19: 虛擬機器的名稱和位置

步驟 6. 選取虛擬機器的目的地儲存區。

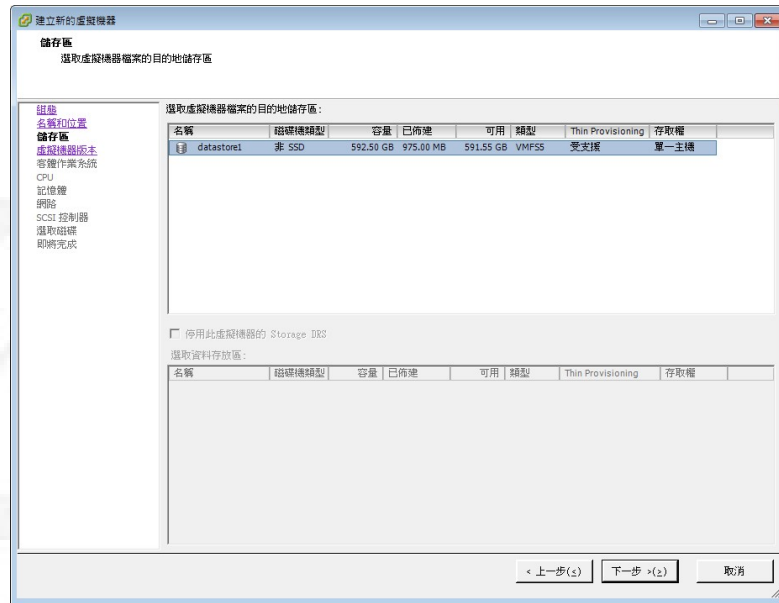


FIGURE B.20: 虛擬機器目的地儲存區

步驟 7. 虛擬機器版本：11

步驟 8. 指定此虛擬機器要使用的客體作業系統。

版本：Ubuntu Linux (64-bit)

步驟 9. 選取虛擬機器的虛擬 CPU 數目：

虛擬通訊端數目：1

每個虛擬通訊端的核心數目：4

步驟 10. 設定虛擬機器的記憶體大小：自訂

步驟 11. 虛擬機器所使用的網路連線。

介面卡：VMXNET3

步驟 12. SCSI 控制器類型：LSI Logic 平行

步驟 13. 選取要使用的磁碟類型：建立新的虛擬磁碟。

步驟 14. 指定虛擬磁碟大小及佈建原則。

步驟 15. 進階選項，不動作，進行下一步。

步驟 16. 點選完成之前編輯虛擬機器設定。

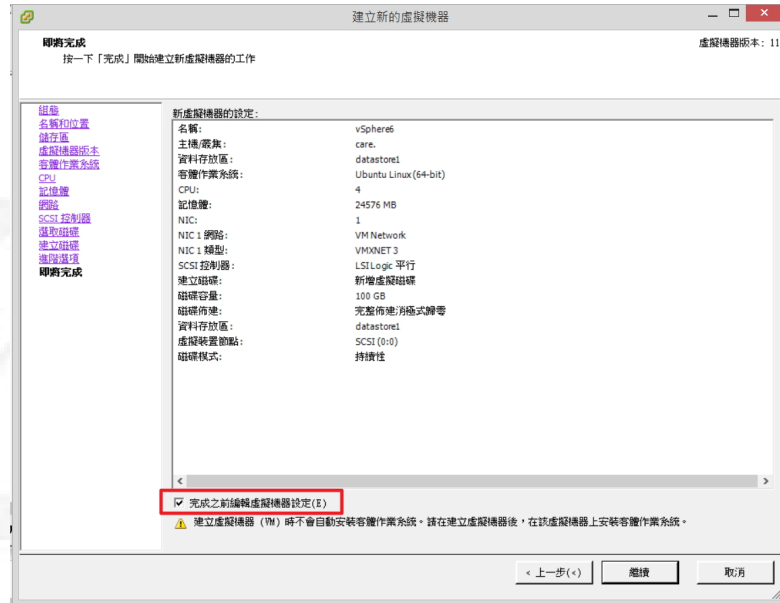


FIGURE B.21: 編輯虛擬機器設定

步驟 17. 設定安裝來源檔。

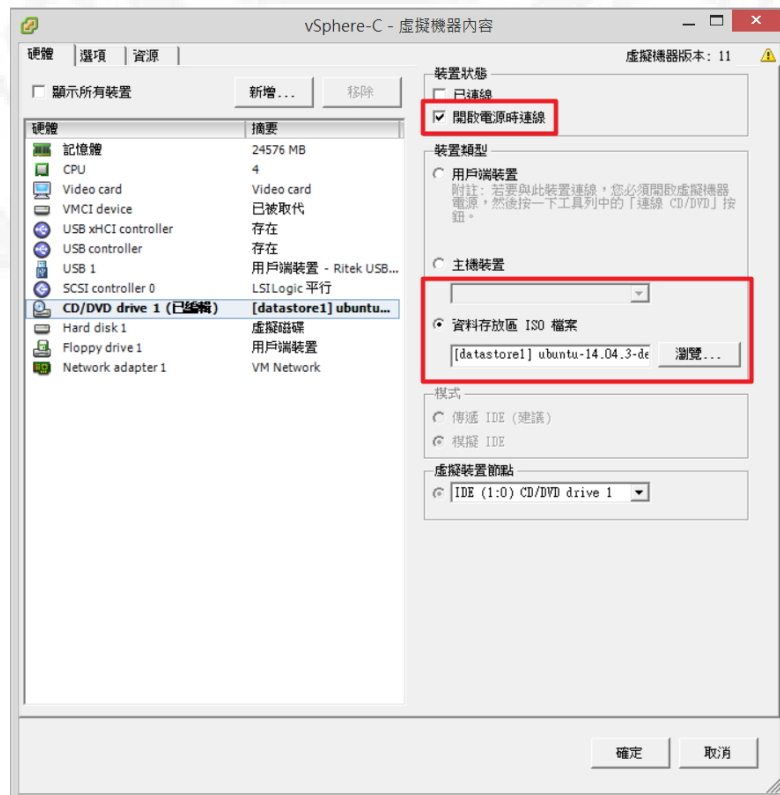


FIGURE B.22: 設定安裝來源檔



步驟 18. 開啟虛擬機器電源。

可切換至主控台看畫面。

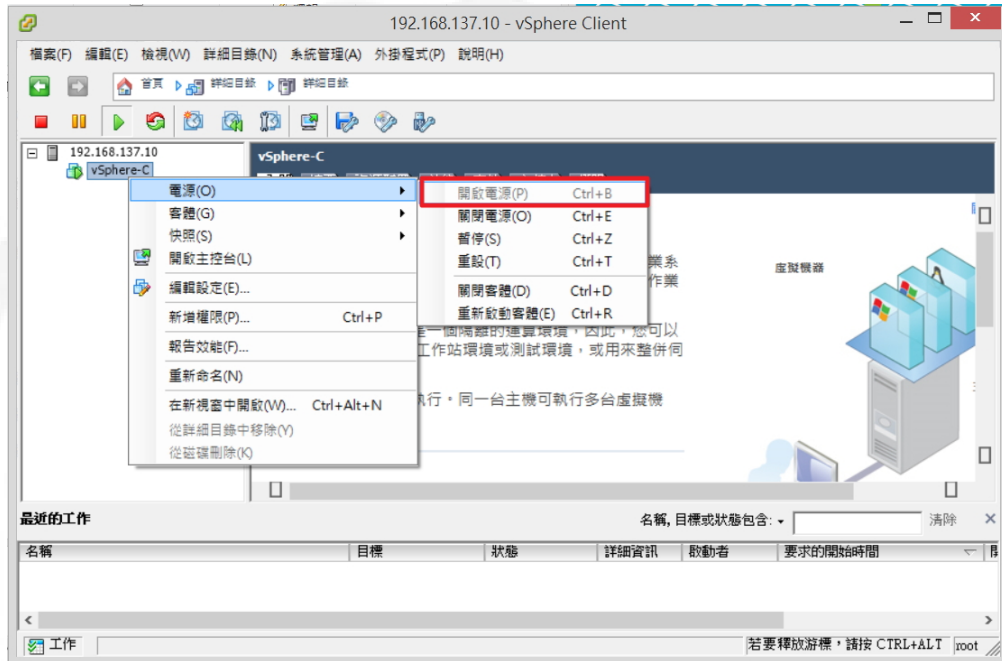


FIGURE B.23: 切換至主控台

步驟 19. 設定 BIOS，使用 CD-ROM 啟動。

- 1) 鍵盤操作：Ctrl+Alt+Insert(Ins)
- 2) F2 進入 BIOS
- 3) Boot 下設定由 CD-ROM Drive 為第一優先。
- 4) F10 儲存與離開
- 5) 重新開啟電源

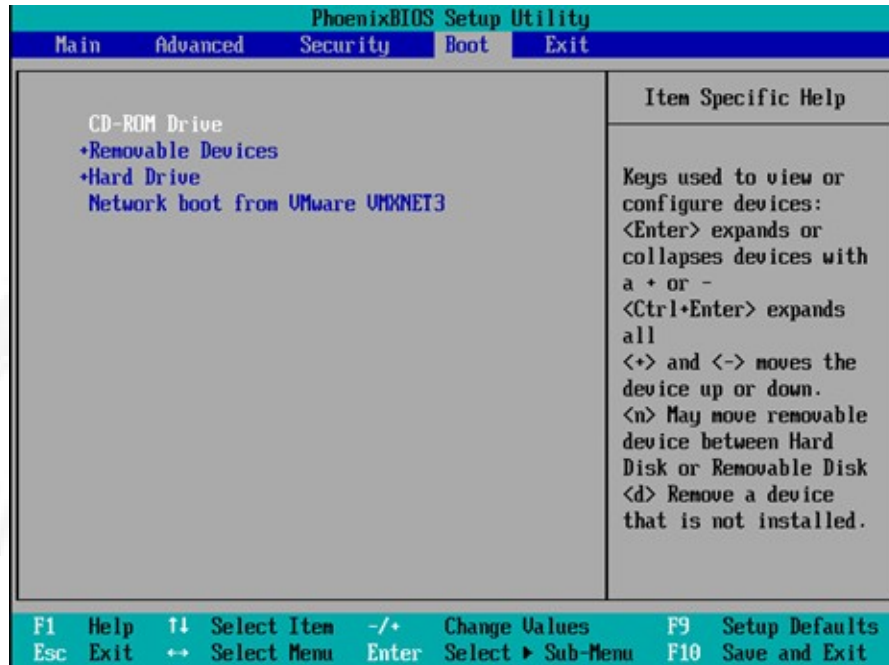


FIGURE B.24: 設定 BIOS

步驟 20. 進入到 Ubuntu 安裝畫面。(後續安裝步驟如附錄 A)

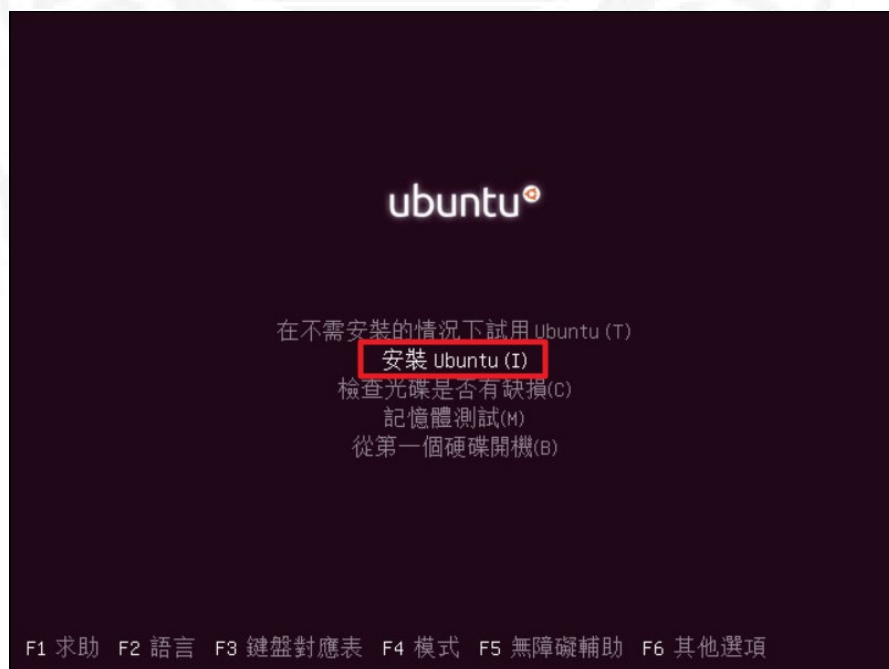


FIGURE B.25: Ubuntu 安裝畫面

步驟 21. 安裝完成時會重新開始，需記得將 BIOS 啟動順序調整回來，使用 HD 開機。

## 附錄 C

# 安裝 VMware Workstation 12

## C.1 於 windows 7 安裝 VMware Workstation 12

步驟 1. 下載 VMware Workstation 後，並執行安裝。

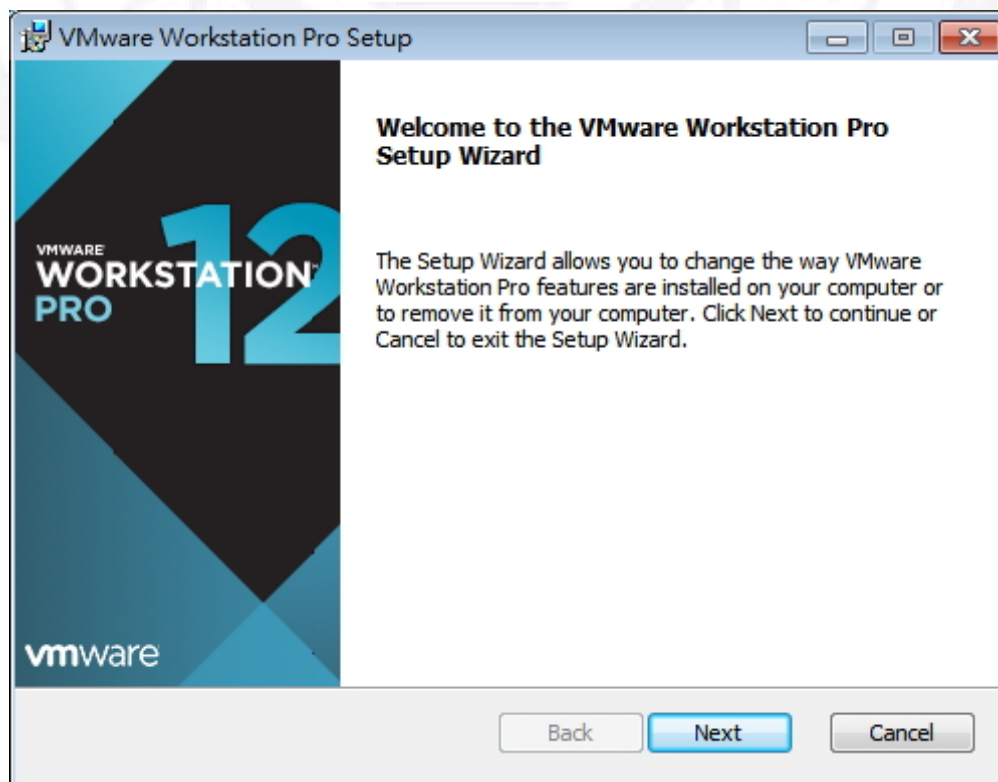


FIGURE C.1: VMware Workstation 安裝畫面

步驟 2. 點選下一步，進行安裝。

步驟 3. 點選同意服務條款後，進行下一步。

步驟 4. 選擇基本 (典型) 安裝或是自訂安裝：在此選擇自訂安裝。

步驟 5. 選擇所需之加選功能後，進行下一步。

步驟 6. 選擇檔案存放位置後，進行下一步。

步驟 7. 選擇在 VMware Workstation 啟動後檢查更新，進行下一步。

步驟 8. 選擇是否回報匿名系統 Value 與使用情況予 VMware 作為參考。

步驟 9. 建立捷徑：自訂。

步驟 10. 點選「continue」進行安裝程序，如需更改上述設定  
請點選 back 回到上述設定處進行更動。

步驟 11. 進行 Install 與設定中，請稍後約四分鐘。

步驟 12. 輸入產品授權碼。

步驟 13. 完成安裝。

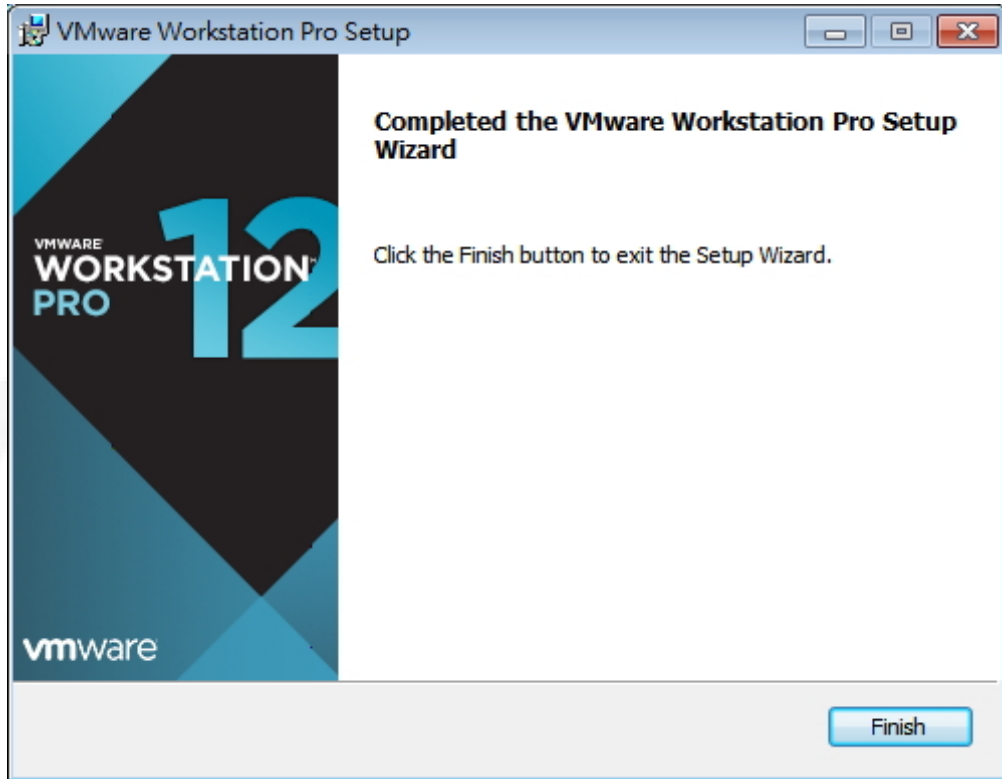


FIGURE C.2: VMware Workstation 完成安裝畫面

步驟 14. 啟動 VMware Workstation。

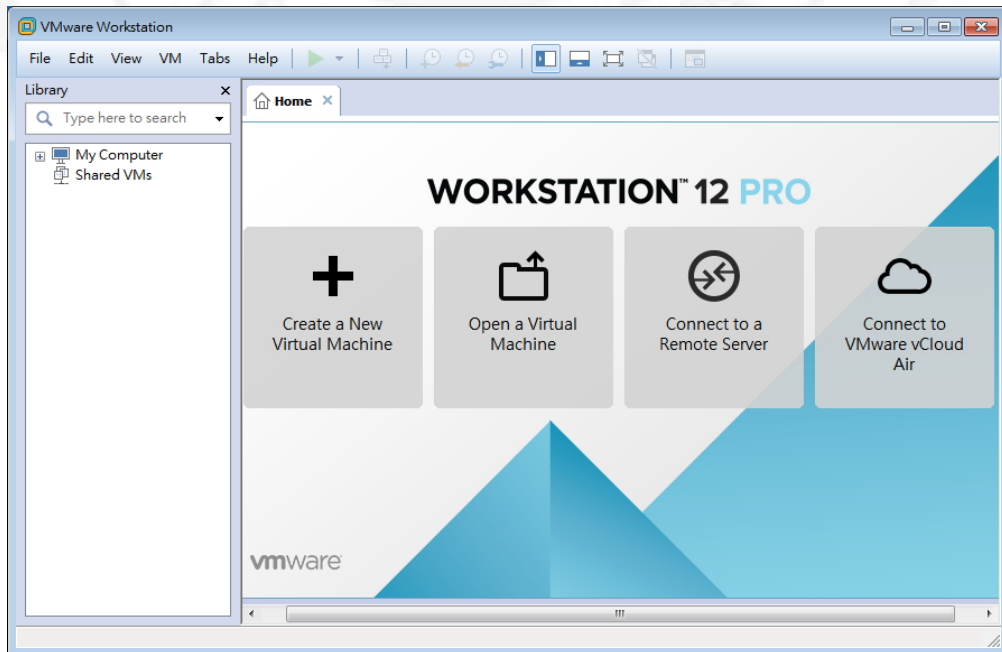


FIGURE C.3: VMware Workstation 啟動畫面

## 附錄 D

### 其他附屬安裝

#### D.1 GCC 編譯器下載安裝

步驟 1. 下載與安裝 gcc。

指令：`sudo apt-get install g++`

步驟 2. 安裝建置必需的檔案。

指令：`sudo apt-get install build-essential`

```
care@ubuntu: ~
care@ubuntu:~$ sudo apt-get install g++
[sudo] password for care:
正在讀取套件清單... 完成
正在重建相依關係
正在讀取狀態資料... 完成
g++ 已是最新版本。
以下套件為自動安裝，並且已經無用：
  linux-headers-3.19.0-25 linux-headers-3.19.0-25-generic
  linux-image-3.19.0-25-generic linux-image-extra-3.19.0-25-generic
Use 'apt-get autoremove' to remove them.
升級 0 個，新安裝 0 個，移除 0 個，有 0 個未被升級。
care@ubuntu:~$ sudo apt-get install build-essential
正在讀取套件清單... 完成
正在重建相依關係
正在讀取狀態資料... 完成
以下套件為自動安裝，並且已經無用：
  linux-headers-3.19.0-25 linux-headers-3.19.0-25-generic
  linux-image-3.19.0-25-generic linux-image-extra-3.19.0-25-generic
Use 'apt-get autoremove' to remove them.
下列的額外套件將被安裝：
  dpkg-dev fakeroot libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libfakeroot
建議套件：
  debian-keyring
下列【新】套件將會被安裝：
  build-essential dpkg-dev fakeroot libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libfakeroot
升級 0 個，新安裝 7 個，移除 0 個，有 0 個未被升級。
需要下載 887 kB 的套件檔。
此操作完成之後，會多佔用 2,436 kB 的磁碟空間。
Do you want to continue? [Y/n] y
```

FIGURE D.1: 安裝建置畫面

步驟 3. 查看是否安裝成功。

指令：g++ -v (也可使用 gcc --version)

```
care@ubuntu: ~
care@ubuntu:~$ g++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.8/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 4.8.4-2ubuntu1~14.04' --with-bugurl=file:///usr/share/doc/gcc-4.8/README.Bugs --enable-languages=c,c++,java,go,d,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.8 --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.8 --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --enable-gnu-unique-object --disable-libmudflap --enable-plugin --with-system-zlib --disable-browser-plugin --enable-java-awt=gtk --enable-gtk-cairo --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-4.8-amd64/jre --enable-java-home --with-jvm-root-dir=/usr/lib/jvm/java-1.5.0-gcj-4.8-amd64 --with-jvm-jar-dir=/usr/lib/jvm-exports/java-1.5.0-gcj-4.8-amd64 --with-arch-directories=amd64 --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --with-tune=generic --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04) ← gcc版本
care@ubuntu:~$
```

FIGURE D.2: 查看安裝狀況畫面



## D.2 OPEN MPI 下載安裝與執行

步驟 1. 下載與安裝。

指令：`wget https://www.open-mpi.org/software/ompi/v1.10/downloads/下載版本.tar.gz`



```
care@ubuntu: ~
care@ubuntu:~$ sudo apt-get install openmpi
正在讀取套件清單... 完成
正在重建相依關係
正在讀取狀態資料... 完成
E: 找不到套件 openmpi
care@ubuntu:~$ wget https://www.open-mpi.org/software/ompi/v1.10/downloads/openm
pi-1.10.0.tar.gz
--2015-10-22 17:46:52-- https://www.open-mpi.org/software/ompi/v1.10/downloads/
openmpi-1.10.0.tar.gz
正在查找主機 www.open-mpi.org (www.open-mpi.org)... 129.79.39.208
正在連接 www.open-mpi.org (www.open-mpi.org)|129.79.39.208|:443... 連上了。
已送出 HTTP 要求，正在等候回應... 200 OK
長度: 19808618 (19M) [application/x-gzip]
Saving to: 'openmpi-1.10.0.tar.gz'
90% [=====> ] 17,833,984 266KB/s eta 7s
```

FIGURE D.3: OPEN MPI 下載安裝畫面

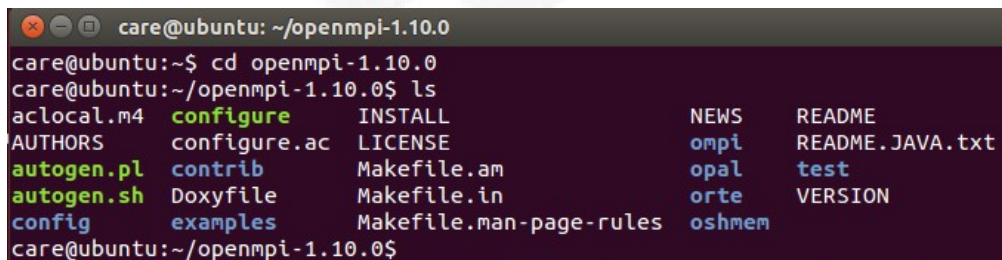
步驟 2. 解壓縮。

指令：`tar xvf openmpi-1.10.0.tar.gz`

步驟 3. 查看其目錄內容。

指令：`cd openmpi-1.10.0`

指令：`ls`



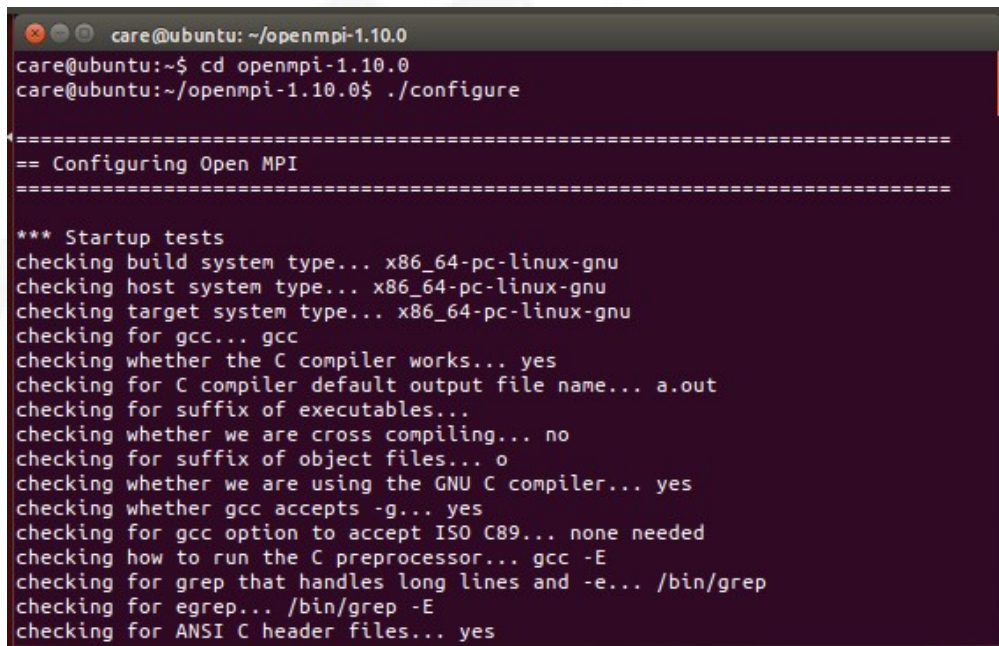
```
care@ubuntu: ~/openmpi-1.10.0
care@ubuntu:~$ cd openmpi-1.10.0
care@ubuntu:~/openmpi-1.10.0$ ls
aclocal.m4  configure          INSTALL          NEWS           README
AUTHORS    configure.ac      LICENSE         ompi           README.JAVA.txt
autogen.pl  contrib          Makefile.am     opal          test
autogen.sh  Doxyfile         Makefile.in     orte          VERSION
config      examples        Makefile.man-page-rules  oshmem
```

FIGURE D.4: 查看目錄內容



步驟 4. 安裝 openMPI(預設至/usr/local/lib)。

指令：`./configure`



```
care@ubuntu: ~/openmpi-1.10.0
care@ubuntu:~/openmpi-1.10.0$ cd openmpi-1.10.0
care@ubuntu:~/openmpi-1.10.0$ ./configure

=====
== Configuring Open MPI
=====

*** Startup tests
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking target system type... x86_64-pc-linux-gnu
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for ANSI C header files... yes
```

FIGURE D.5: 安裝 openMPI

步驟 5. 安裝所有需要的文件。

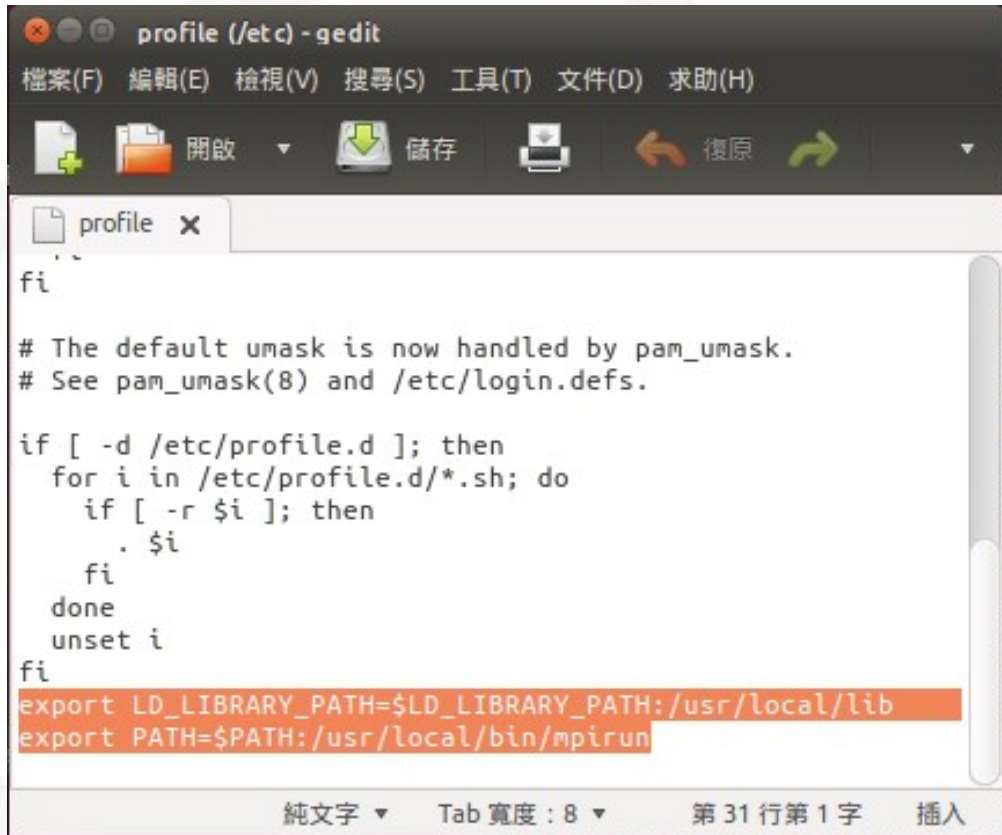
指令：`sudo make all install`

輸入密碼：`****`

步驟 6. 添加共享路徑。

指令：`sudo gedit /etc/profile`

在最後一行加入環境變數：

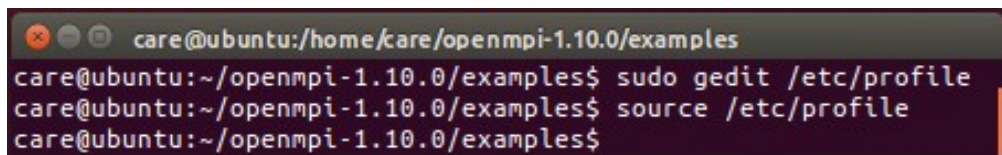


```
profile (/etc) - gedit
檔案(F) 編輯(E) 檢視(V) 搜尋(S) 工具(T) 文件(D) 求助(H)
+ 開啟 儲存 復原
profile x
fi
# The default umask is now handled by pam_umask.
# See pam_umask(8) and /etc/login.defs.
if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
export PATH=$PATH:/usr/local/bin/mpirun
純文字 Tab 寬度: 8 第 31 行第 1 字 插入
```

FIGURE D.6: 環境變數

步驟 7. 使配置文件生效。

指令：`source /etc/profile`



```
care@ubuntu:/home/care/openmpi-1.10.0/examples
care@ubuntu:~/openmpi-1.10.0/examples$ sudo gedit /etc/profile
care@ubuntu:~/openmpi-1.10.0/examples$ source /etc/profile
care@ubuntu:~/openmpi-1.10.0/examples$
```

FIGURE D.7: 配置生效

步驟 8. 執行測試是否安裝成功 (啟用範例)。

```
care@ubuntu:/home/care/openmpi-1.10.0/examples
care@ubuntu:~/openmpi-1.10.0$ cd
care@ubuntu:~$ cd openmpi-1.10.0/examples
care@ubuntu:~/openmpi-1.10.0/examples$ make
mpicc -g hello_c.c -o hello_c
mpicc -g ring_c.c -o ring_c
mpicc -g connectivity_c.c -o connectivity_c
make[1]: Entering directory `/home/care/openmpi-1.10.0/examples'
make[2]: Entering directory `/home/care/openmpi-1.10.0/examples'
mpicc -g hello_cxx.c -o hello_cxx
mpicc -g ring_cxx.c -o ring_cxx
make[2]: Leaving directory `/home/care/openmpi-1.10.0/examples'
make[1]: Leaving directory `/home/care/openmpi-1.10.0/examples'
make[1]: Entering directory `/home/care/openmpi-1.10.0/examples'
make[2]: Entering directory `/home/care/openmpi-1.10.0/examples'
shmemcc -g hello_oshmem.c -o hello_oshmem
make[2]: Leaving directory `/home/care/openmpi-1.10.0/examples'
make[2]: Entering directory `/home/care/openmpi-1.10.0/examples'
shmemcc -g ring_oshmem.c -o ring_oshmem
make[2]: Leaving directory `/home/care/openmpi-1.10.0/examples'
make[2]: Entering directory `/home/care/openmpi-1.10.0/examples'
shmemcc -g oshmem_shmalloc.c -o oshmem_shmalloc
make[2]: Leaving directory `/home/care/openmpi-1.10.0/examples'
make[2]: Entering directory `/home/care/openmpi-1.10.0/examples'
shmemcc -g oshmem_circular_shift.c -o oshmem_circular_shift
make[2]: Leaving directory `/home/care/openmpi-1.10.0/examples'
make[2]: Entering directory `/home/care/openmpi-1.10.0/examples'
shmemcc -g oshmem_max_reduction.c -o oshmem_max_reduction
make[2]: Leaving directory `/home/care/openmpi-1.10.0/examples'
make[2]: Entering directory `/home/care/openmpi-1.10.0/examples'
shmemcc -g oshmem_strided_puts.c -o oshmem_strided_puts
make[2]: Leaving directory `/home/care/openmpi-1.10.0/examples'
make[2]: Entering directory `/home/care/openmpi-1.10.0/examples'
shmemcc -g oshmem_symmetric_data.c -o oshmem_symmetric_data
make[2]: Leaving directory `/home/care/openmpi-1.10.0/examples'
make[1]: Leaving directory `/home/care/openmpi-1.10.0/examples'
care@ubuntu:~/openmpi-1.10.0/examples$ mpirun -np 4 hello_c
Hello, world, I am 0 of 4, (Open MPI v1.10, package: Open MPI care@ubuntu Distri
bution, ident: 1.10.0, repo rev: v1.10-dev-293-gf694355, Aug 24, 2015, 122)
Hello, world, I am 1 of 4, (Open MPI v1.10, package: Open MPI care@ubuntu Distri
bution, ident: 1.10.0, repo rev: v1.10-dev-293-gf694355, Aug 24, 2015, 122)
Hello, world, I am 2 of 4, (Open MPI v1.10, package: Open MPI care@ubuntu Distri
bution, ident: 1.10.0, repo rev: v1.10-dev-293-gf694355, Aug 24, 2015, 122)
Hello, world, I am 3 of 4, (Open MPI v1.10, package: Open MPI care@ubuntu Distri
bution, ident: 1.10.0, repo rev: v1.10-dev-293-gf694355, Aug 24, 2015, 122)
care@ubuntu:~/openmpi-1.10.0/examples$
```

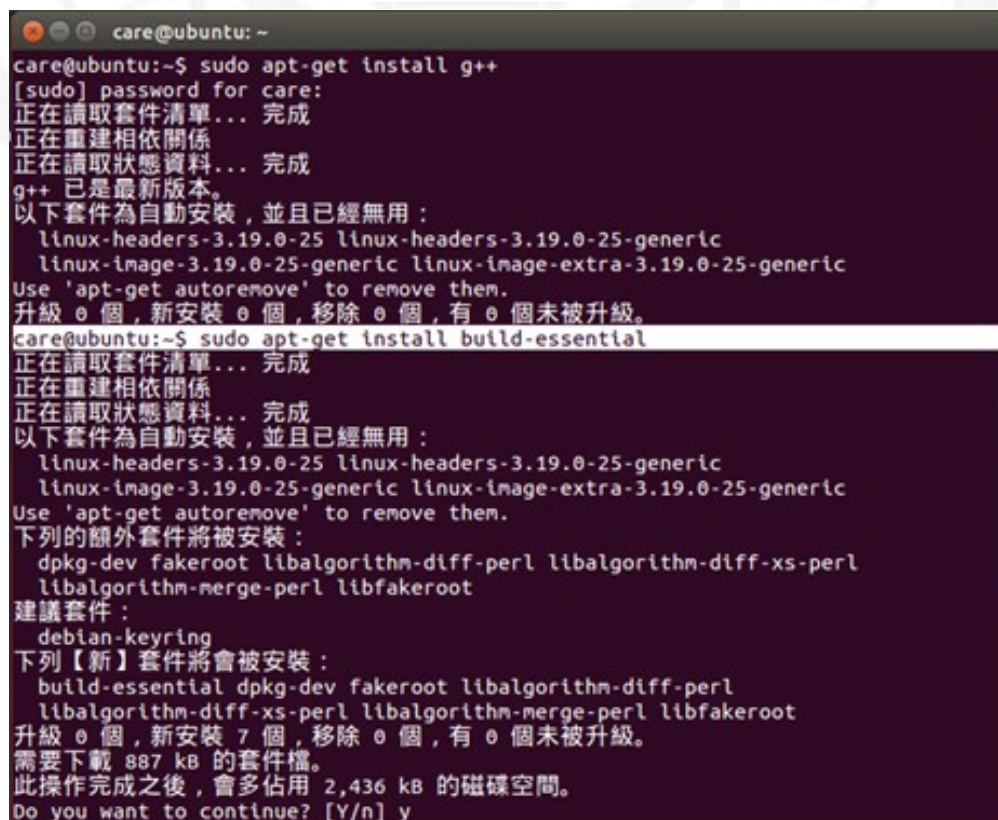
FIGURE D.8: 執行測試 (啟用範例)

## 附錄 E

# 效能測試安裝與執行

## E.1 HPL 安裝與實測

步驟 1. 下載與安裝 GCC。



```
care@ubuntu: ~  
care@ubuntu:~$ sudo apt-get install g++  
[sudo] password for care:  
正在讀取套件清單... 完成  
正在重建相依關係  
正在讀取狀態資料... 完成  
g++ 已是最新版本。  
以下套件為自動安裝，並且已經無用：  
  linux-headers-3.19.0-25 linux-headers-3.19.0-25-generic  
  linux-image-3.19.0-25-generic linux-image-extra-3.19.0-25-generic  
Use 'apt-get autoremove' to remove them.  
升級 0 個，新安裝 0 個，移除 0 個，有 0 個未被升級。  
care@ubuntu:~$ sudo apt-get install build-essential  
正在讀取套件清單... 完成  
正在重建相依關係  
正在讀取狀態資料... 完成  
以下套件為自動安裝，並且已經無用：  
  linux-headers-3.19.0-25 linux-headers-3.19.0-25-generic  
  linux-image-3.19.0-25-generic linux-image-extra-3.19.0-25-generic  
Use 'apt-get autoremove' to remove them.  
下列的額外套件將被安裝：  
  dpkg-dev fakeroot libalgorithm-diff-perl libalgorithm-diff-xs-perl  
  libalgorithm-merge-perl libfakeroot  
建議套件：  
  debian-keyring  
下列【新】套件將會被安裝：  
  build-essential dpkg-dev fakeroot libalgorithm-diff-perl  
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libfakeroot  
升級 0 個，新安裝 7 個，移除 0 個，有 0 個未被升級。  
需要下載 887 kB 的套件檔。  
此操作完成之後，會多佔用 2,436 kB 的磁碟空間。  
Do you want to continue? [Y/n] y
```

FIGURE E.1: 下載與安裝 GCC



步驟 2. 檢查安裝狀態。

```
care@ubuntu: ~  
care@ubuntu:~$ g++ -v  
Using built-in specs.  
COLLECT_GCC=g++  
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.8/lto-wrapper  
Target: x86_64-linux-gnu  
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 4.8.4-2ubuntu1~14.04' --with-bugurl=file:///usr/share/doc/gcc-4.8/README.Bugs --enable-languages=c,c++,java,go,d,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.8 --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.8 --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --enable-gnu-unique-object --disable-libmudflap --enable-plugin --with-system-zlib --disable-browser-plugin --enable-java-awt=gtk --enable-gtk-cairo --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-4.8-amd64/jre --enable-java-home --with-jvm-root-dir=/usr/lib/jvm/java-1.5.0-gcj-4.8-amd64 --with-jvm-jar-dir=/usr/lib/jvm-exports/java-1.5.0-gcj-4.8-amd64 --with-arch-directory=amd64 --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --with-tune=generic --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu  
Thread model: posix  
gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04) ← gcc版本  
care@ubuntu:~$
```

FIGURE E.2: 檢查安裝狀態

步驟 3.HPL scripts 批次檔下載。

```
care@ubuntu: ~  
care@ubuntu:~$ wget http://ctyang.thu.edu.tw/install_hpcc_hpl_03082016.sh  
--2016-03-13 23:48:01-- http://ctyang.thu.edu.tw/install_hpcc_hpl_03082016.sh  
正在查找主機 ctyang.thu.edu.tw (ctyang.thu.edu.tw)... 140.128.99.228  
正在連接 ctyang.thu.edu.tw (ctyang.thu.edu.tw)|140.128.99.228|:80... 連上了。  
已送出 HTTP 要求, 正在等候回應... 200 OK  
長度: 621 [application/x-sh]  
Saving to: 'install_hpcc_hpl_03082016.sh'  
  
100%[=====] 621 --.-K/s in 0s  
  
2016-03-13 23:48:02 (104 MB/s) - 'install_hpcc_hpl_03082016.sh' saved [621/621]  
care@ubuntu:~$
```

FIGURE E.3: HPL scripts 批次檔下載

步驟 4. 使用 GCC Compiler 編譯函式庫 Blas 轉換為 xhpl(機器語言)的過程，並且計算 HPL 測試基準值與顯示測試所花費的時間。

```
care@ubuntu: ~
care@ubuntu:~$ time sh install_hpcc_hpl_03082016.sh ubuntu 64 4
--2016-03-13 23:49:48-- http://ctyang.thu.edu.tw/sh_hpl_hpcc_N0source_03082016.tar.gz
正在查找主機 ctyang.thu.edu.tw (ctyang.thu.edu.tw)... 140.128.99.228
正在連接 ctyang.thu.edu.tw (ctyang.thu.edu.tw)[140.128.99.228]:80... 連上了。
已送出 HTTP 要求，正在等候回應... 200 OK
長度: 55771 (54K) [application/x-gzip]
Saving to: 'sh_hpl_hpcc_N0source_03082016.tar.gz'

100%[=====] 55,771 362KB/s in 0.2s

2016-03-13 23:49:48 (362 KB/s) - 'sh_hpl_hpcc_N0source_03082016.tar.gz' saved [55771/55771]

--2016-03-13 23:49:48-- http://ctyang.thu.edu.tw/install_ubuntu_03082016.sh
正在查找主機 ctyang.thu.edu.tw (ctyang.thu.edu.tw)... 140.128.99.228
正在連接 ctyang.thu.edu.tw (ctyang.thu.edu.tw)[140.128.99.228]:80... 連上了。
已送出 HTTP 要求，正在等候回應... 200 OK
```

FIGURE E.4: GCC Compiler 編譯函式庫過程

步驟 5. 取得效能基準。

```
care@ubuntu: ~
care@ubuntu:~$ more ~/hpl_hpcc_gcc_open_goto/hpl-2.2/bin/Linux_goto/HPL.out
=====
HPLinpack 2.2 -- High-Performance Linpack benchmark -- February 24, 2016
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
=====

An explanation of the input/output parameters follows:
T/V : Wall time / encoded variant.
N : The order of the coefficient matrix A.
NB : The partitioning blocking factor.
P : The number of process rows.
Q : The number of process columns.
Time : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N : 10000
NB : 88 128 108 168
PMAP : Row-major process mapping
P : 2
Q : 2
PFACT : Left Crout Right
NBMIN : 4
NDIV : 2
RFACT : Crout
BCAST : 1ring
DEPTH : 0
SWAP : Mix (threshold = 64)
L1 : transposed form
U : transposed form
EQUIL : yes
ALIGN : 8 double precision words
=====
```

FIGURE E.5: 取得效能基準

## E.2 GotoBlas 與 OpenBlas 測試

### E.2.1 GCC 編譯器執行測試

步驟 1. 執行 GCC 編譯 GotoBlas 需先指定核心數，並指定輸出數據。

```
care@ubuntu: ~/hpl_hpcc gcc open_goto/hpl-2.2/bin/Linux_goto
care@ubuntu:~$ cd ~/hpl_hpcc gcc open_goto/hpl-2.2/bin/Linux_goto/
care@ubuntu:~/hpl_hpcc gcc open_goto/hpl-2.2/bin/Linux_goto$ mpirun -np 4 ./xhpl
care@ubuntu:~/hpl_hpcc gcc open_goto/hpl-2.2/bin/Linux_goto$ more HPL.out
=====
HPLinpack 2.2 -- High-Performance Linpack benchmark -- February 24, 2016
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
=====

An explanation of the input/output parameters follows:
T/V : Wall time / encoded variant.
N : The order of the coefficient matrix A.
NB : The partitioning blocking factor.
P : The number of process rows.
Q : The number of process columns.
Time : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N : 10000
NB : 88 128 108 168
PMAP : Row-major process mapping
P : 2
Q : 2
PFACT : Left Crout Right
NBMIN : 4
NDIV : 2
RFACT : Crout
BCAST : 1ring
--更多--(12%)
```

FIGURE E.6: 執行 GCC 編譯 GotoBlas 畫面

步驟 2. 執行 GCC 編譯 OpenBlas 需先指定線程數與核心數，並指定輸出數據。

```
care@ubuntu: ~/hpl_hpcc gcc open_goto/hpl-2.2/bin/Linux_open
care@ubuntu:~$ cd ~/hpl_hpcc gcc open_goto/hpl-2.2/bin/Linux_open/
care@ubuntu:~/hpl_hpcc gcc open_goto/hpl-2.2/bin/Linux_open$ export OMP_NUM_THREADS=1
care@ubuntu:~/hpl_hpcc gcc open_goto/hpl-2.2/bin/Linux_open$ mpirun -np 4 ./xhpl
care@ubuntu:~/hpl_hpcc gcc open_goto/hpl-2.2/bin/Linux_open$ more HPL.out
=====
HPLinpack 2.2 -- High-Performance Linpack benchmark -- February 24, 2016
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
=====

An explanation of the input/output parameters follows:
T/V : Wall time / encoded variant.
N : The order of the coefficient matrix A.
NB : The partitioning blocking factor.
P : The number of process rows.
Q : The number of process columns.
Time : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N : 10000
NB : 88 128 108 168
PMAP : Row-major process mapping
P : 2
Q : 2
--更多--(10%)
```

FIGURE E.7: 執行 GCC 編譯 OpenBlas 畫面

## E.2.2 GCC 執行 Intel MKL 測試

本測試前需先進行 ICC 編譯器安裝（請見下一小節），並引入環境變數。

步驟 1. 執行 GCC 編譯 Intel MKL 需先代入環境變數與指定核心數，  
並指定輸出數據。

```
care@ubuntu: ~/hpl_hpcc_gcc_mkl/hpl-2.2/bin/Linux_gcc_mkl
care@ubuntu:~$ which mpirun
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpirun
care@ubuntu:~$ cd ~/hpl_hpcc_gcc_mkl/hpl-2.2/bin/Linux_gcc_mkl/
care@ubuntu:~/hpl_hpcc_gcc_mkl/hpl-2.2/bin/Linux_gcc_mkl$ /usr/bin/mpirun -np 4 ./xhpl
care@ubuntu:~/hpl_hpcc_gcc_mkl/hpl-2.2/bin/Linux_gcc_mkl$ more HPL.out
=====
HPLinpack 2.2 -- High-Performance Linpack benchmark -- February 24, 2016
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
=====
An explanation of the input/output parameters follows:
T/V   : Wall time / encoded variant.
N     : The order of the coefficient matrix A.
NB    : The partitioning blocking factor.
P     : The number of process rows.
Q     : The number of process columns.
Time  : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      : 10000
NB     : 88      128      108      168
PMAP   : Row-major process mapping
P      : 2
Q      : 2
PFACT  : Left      Crout    Right
NBMIN  : 4
NDIV   : 2
RFACT  : Crout
BCAST  : iring
DEPTH  : 0
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
--更多--(13%)
```

FIGURE E.8: 執行 GCC 編譯 Intel MKL 畫面

## E.2.3 ICC 編譯器安裝與測試

步驟 1. 實驗進行前先進行 ICC 編譯器安裝與執行

（Intel Parallel Studio XE 2016），請先至 Intel 官網  
申請下載，需輸入個人資料並至您留下的聯絡信箱內  
讀取授權碼。[https://software.intel.com/en-us/  
intel-parallel-studio-xe](https://software.intel.com/en-us/intel-parallel-studio-xe)



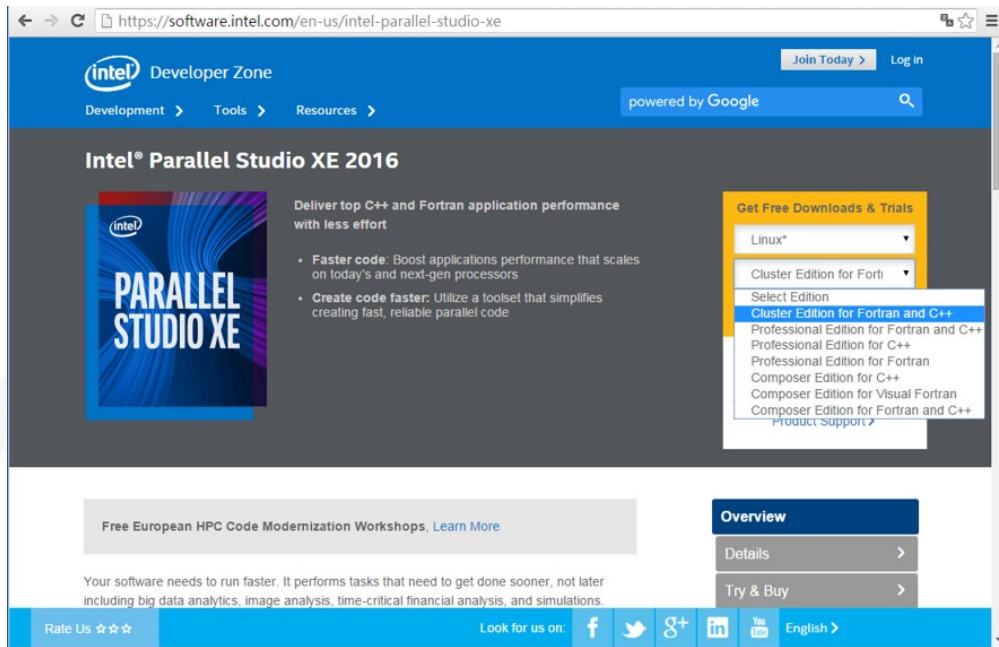


FIGURE E.9: Intel 官網申請畫面

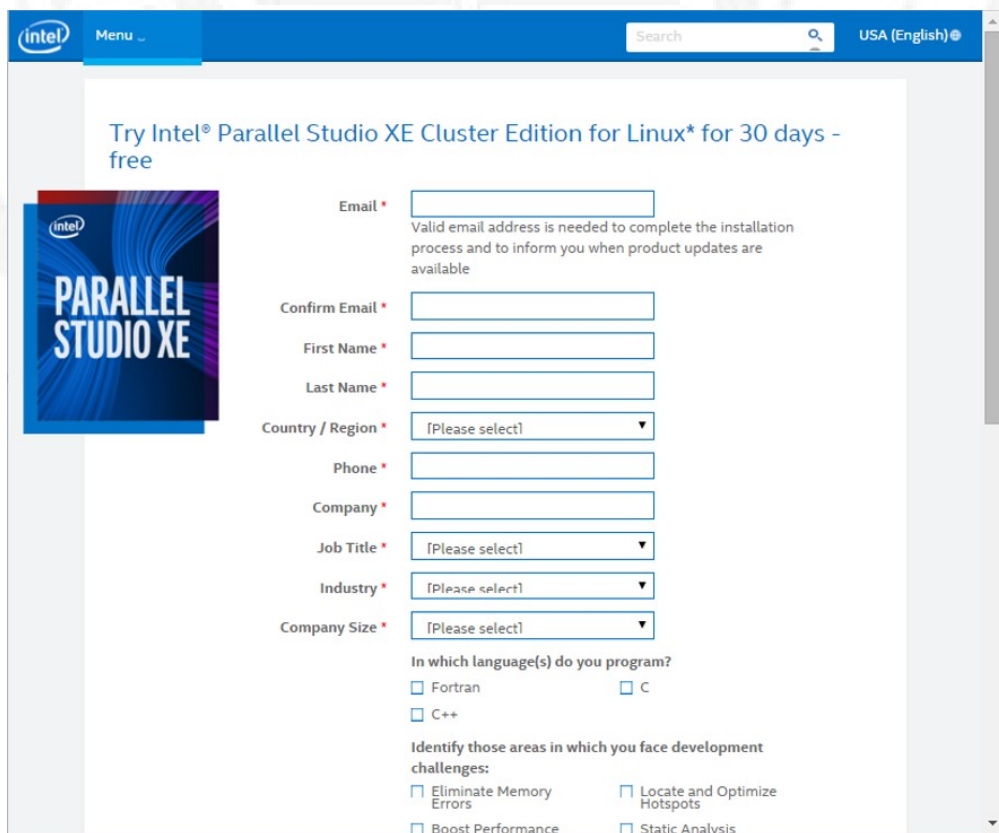


FIGURE E.10: Intel 官網資料登錄畫面

步驟 2. 執行 ICC 編譯 GotoBlas 需先指定核心數，並指定輸出數據。

```
root@ubuntu: /home/care/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto
care@ubuntu:~$ cd ~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto/
care@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto$ sudo su
[sudo] password for care:
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# source /opt/intel/compil
ers_and_libraries_2016.2.181/linux/bin/iccvars.sh intel64
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# source /opt/intel/compil
ers_and_libraries_2016.2.181/linux/bin/compilervars.sh intel64
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# source /opt/intel/compil
ers_and_libraries_2016.2.181/linux/bin/ifortvars.sh intel64
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# source /opt/intel/impi/5
.1.3.181/bin64/mpivars.sh intel64
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# which icc
/opt/intel/compilers_and_libraries_2016.2.181/linux/bin/intel64/icc
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# which mpicc
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpicc
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# which ifort
/opt/intel/compilers_and_libraries_2016.2.181/linux/bin/intel64/ifort
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# which mpiicc
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpiicc
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# which mpirun
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpirun
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# mpirun -np 4 ./xhpl
root@ubuntu:~/hpl_hpcc_icc_open_goto/hpl-2.2/bin/Linux_goto# more HPL.out
=====
HPLinpack 2.2 -- High-Performance Linpack benchmark -- February 24, 2016
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
=====

An explanation of the input/output parameters follows:
T/V      : Wall time / encoded variant.
N        : The order of the coefficient matrix A.
NB       : The partitioning blocking factor.
P        : The number of process rows.
Q        : The number of process columns.
Time     : Time in seconds to solve the linear system.
Gflops   : Rate of execution for solving the linear system.

The following parameter values will be used:

N        : 10000
NB       : 88      128      108      168
PMAP     : Row-major process mapping
P        : 2
Q        : 2
PFACT   : Left    Crout    Right
NBMIN   : 4
NDIV    : 2
RFACT   : Crout
BCAST   : 1ring
```

FIGURE E.11: 執行 ICC 編譯 GotoBlas 畫面

步驟 3. 執行 ICC 編譯 OpenBlas 需先指定線程數與核心數，並指定輸出數據。

```
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open
care@ubuntu:~$ cd ~/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open/
care@ubuntu:~/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open$ sudo su
[sudo] password for care:
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# source /opt/intel/compilers_and_libraries_2016.2.181/linux/bin/ic
cvars.sh intel64
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# source /opt/intel/compilers_and_libraries_2016.2.181/linux/bin/cc
mpitlvars.sh intel64
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# source /opt/intel/compilers_and_libraries_2016.2.181/linux/bin/if
ortvars.sh intel64
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# source /opt/intel/impi/5.1.3.181/bin64/mpivars.sh intel64
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# which icc
/opt/intel/compilers_and_libraries_2016.2.181/linux/bin/intel64/icc
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# which mpicc
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpicc
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# which ifort
/opt/intel/compilers_and_libraries_2016.2.181/linux/bin/intel64/ifort
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# which mpiicc
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpiicc
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# which mpirun
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpirun
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# export OMP_NUM_THREADS=1
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# mpirun -np 4 ./xhpl
root@ubuntu: /home/care/hpl_hgcc_icc_open_goto/hpl-2.2/bin/Linux_open# more HPL.out
=====
HPLinpack 2.2 -- High-Performance Linpack benchmark -- February 24, 2016
Written by A. Pettit and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
=====
An explanation of the input/output parameters follows:
T/V : Wall time / encoded variant.
N : The order of the coefficient matrix A.
NB : The partitioning blocking factor.
P : The number of process rows.
Q : The number of process columns.
Time : Time in seconds to solve the linear system.
GFlops : Rate of execution for solving the linear system.

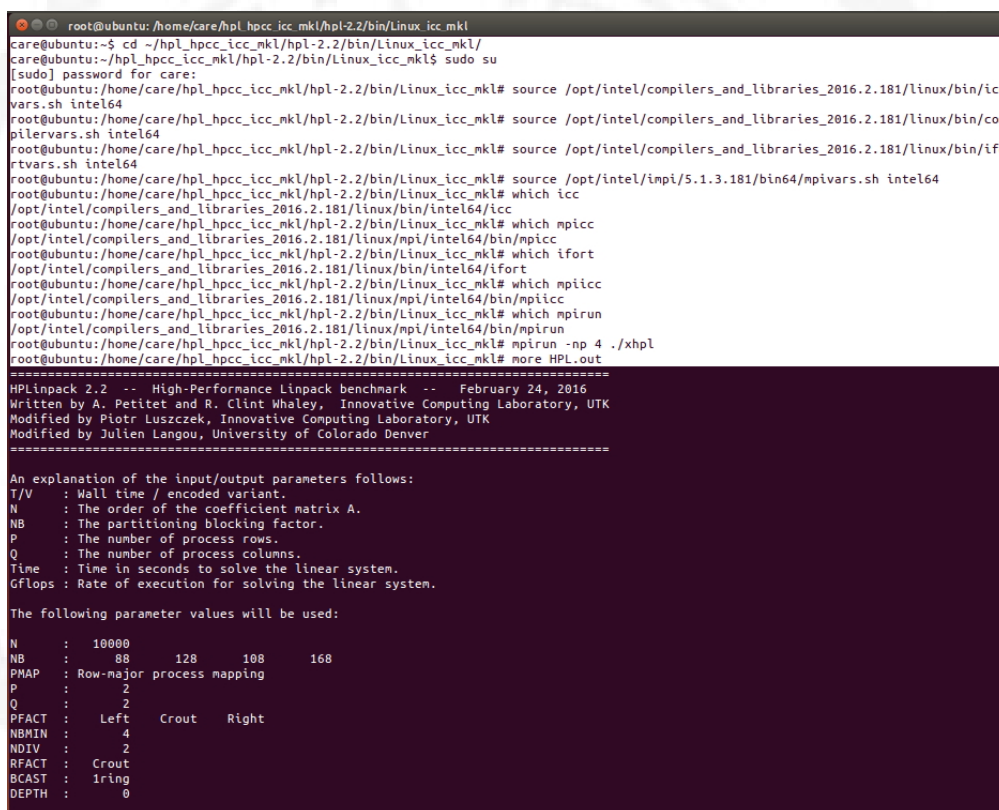
The following parameter values will be used:
N : 10000
NB : 88 128 108 168
PMAP : Row-major process mapping
P : 2
Q : 2
PFACT : Left Crout Right
NBMIN : 4
NDIV : 2
RFACT : Crout
```

FIGURE E.12: 執行 ICC 編譯 OpenBlas 畫面

## E.2.4 ICC 執行 Intel MKL 測試

步驟 1. 執行 ICC 編譯 Intel MKL 需先代入環境變數與指定核心數，  
並指定輸出數據。

步驟 2. 執行 ICC 編譯 Intel MKL。



```
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl
care@ubuntu:~$ cd ~/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl/
care@ubuntu:~/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl$ sudo su
[sudo] password for care:
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# source /opt/intel/compilers_and_libraries_2016.2.181/linux/bin/icc
vars.sh intel64
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# source /opt/intel/compilers_and_libraries_2016.2.181/linux/bin/com
pilervars.sh intel64
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# source /opt/intel/compilers_and_libraries_2016.2.181/linux/bin/ifo
rtvars.sh intel64
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# source /opt/intel/mpi/5.1.3.181/bin64/mpivars.sh intel64
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# which icc
/opt/intel/compilers_and_libraries_2016.2.181/linux/bin/intel64/icc
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# which mpicc
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpicc
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# which ifort
/opt/intel/compilers_and_libraries_2016.2.181/linux/bin/intel64/ifort
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# which mpiicc
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpicc
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# which mpirun
/opt/intel/compilers_and_libraries_2016.2.181/linux/mpi/intel64/bin/mpirun
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# mpirun -np 4 ./xhpl
root@ubuntu: /home/care/hpl_hpcc_icc_mkl/hpl-2.2/bin/Linux_icc_mkl# more HPL.out
=====
HPLinpack 2.2 -- High-Performance Linpack benchmark -- February 24, 2016
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
=====
An explanation of the input/output parameters follows:
T/V : Wall time / encoded variant.
N : The order of the coefficient matrix A.
NB : The partitioning blocking factor.
P : The number of process rows.
Q : The number of process columns.
Time : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:
N : 10000
NB : 88 128 108 168
PMAP : Row-major process mapping
P : 2
Q : 2
PFACT : Left Crout Right
NBRIN : 4
NDIV : 2
RFACT : Crout
BCAST : iring
DEPTH : 0
```

FIGURE E.13: 執行 ICC 編譯 Intel MKL 畫面

## 附錄 F

### 測試數據統計表

#### F.1 HPL 統計表

E3-HPL						
架構	函式庫	網格	分區分塊	浮點數	單位	效能比
A	GOTO	10000	108	47.58	e+01	22.53%
	OPEN	10000	128	124.30	e+02	58.85%
B	GOTO	10000	108	48.59	e+01	23.01%
	OPEN	10000	128	120.50	e+02	57.05%
C	GOTO	10000	108	48.86	e+01	23.13%
	OPEN	10000	128	127.30	e+02	60.27%
D	GOTO	10000	88	48.32	e+01	22.88%
	OPEN	10000	128	128.70	e+02	60.94%

FIGURE F.1: HPL 數據統計-E3

i7-HPL						
架構	函式庫	網格	分區分塊	浮點數	單位	效能比
<b>A</b>	GOTO	10000	108	63.03	e+01	39.79%
	OPEN	10000	128	106.90	e+02	67.49%
<b>B</b>	GOTO	10000	108	65.03	e+01	41.05%
	OPEN	10000	88	111.60	e+02	70.45%
<b>C</b>	GOTO	10000	108	66.83	e+01	42.19%
	OPEN	10000	88	117.70	e+02	74.31%
<b>D</b>	GOTO	10000	88	67.58	e+01	42.66%
	OPEN	10000	88	116.50	e+02	73.55%

FIGURE F.2: HPL 數據統計-i7

## F.2 GCC 統計表

E3-GCC						
架構	函式庫	網格	分區分塊	浮點數	單位	效能比
<b>A</b>	GOTO	10000	108	47.32	e+01	22.41%
	OPEN	10000	128	123.90	e+02	58.66%
	MKL	10000	128	134.60	e+02	63.73%
	GOTO	20000	128	49.76	e+01	23.56%
	OPEN	20000	128	136.10	e+02	64.44%
	MKL	20000	128	149.50	e+02	70.79%
	GOTO	30000	128	50.61	e+01	23.96%
	OPEN	30000	128	148.70	e+02	70.41%
	MKL	30000	168	162.60	e+02	76.99%
<b>B</b>	GOTO	10000	108	48.41	e+01	22.92%
	OPEN	10000	128	127.20	e+02	60.23%
	MKL	10000	128	137.40	e+02	65.06%
	GOTO	20000	128	50.61	e+01	23.96%
	OPEN	20000	128	142.70	e+02	67.57%
	MKL	20000	128	153.40	e+02	72.63%
	GOTO	30000	168	51.73	e+01	24.49%
	OPEN	30000	128	152.40	e+02	72.16%
	MKL	30000	168	169.20	e+02	80.11%
<b>C</b>	GOTO	10000	108	48.74	e+01	23.08%
	OPEN	10000	128	127.30	e+02	60.27%
	MKL	10000	168	144.60	e+02	68.47%
	GOTO	20000	108	51.09	e+01	24.19%
	OPEN	20000	128	146.50	e+02	69.37%
	MKL	20000	168	163.10	e+02	77.23%
	GOTO	30000	128	52.09	e+01	24.66%
	OPEN	30000	128	156.80	e+02	74.24%
	MKL	30000	168	171.50	e+02	81.20%
<b>D</b>	GOTO	10000	108	47.99	e+01	22.72%
	OPEN	10000	128	129.20	e+02	61.17%
	MKL	10000	168	142.90	e+02	67.66%
	GOTO	20000	168	50.60	e+01	23.96%
	OPEN	20000	128	145.10	e+02	68.70%
	MKL	20000	168	162.90	e+02	77.13%
	GOTO	30000	168	51.60	e+01	24.43%
	OPEN	30000	128	155.90	e+02	73.82%
	MKL	30000	168	172.30	e+02	81.58%

FIGURE F.3: GCC 數據統計-E3



i7-GCC						
架構	函式庫	網格	分區分塊	浮點數	單位	效能比
<b>A</b>	GOTO	10000	108	62.27	e+01	39.31%
	OPEN	10000	108	107.70	e+02	67.99%
	MKL	10000	168	100.40	e+02	63.38%
	GOTO	20000	108	67.09	e+01	42.35%
	OPEN	20000	128	118.80	e+02	75.00%
	MKL	20000	168	108.80	e+02	68.69%
	GOTO	30000	168	68.96	e+01	43.54%
	OPEN	30000	168	125.10	e+02	78.98%
	MKL	30000	168	111.20	e+02	70.20%
<b>B</b>	GOTO	10000	88	64.96	e+01	41.01%
	OPEN	10000	88	111.10	e+02	70.14%
	MKL	10000	168	102.20	e+02	64.52%
	GOTO	20000	108	68.32	e+01	43.13%
	OPEN	20000	128	122.60	e+02	77.40%
	MKL	20000	168	111.50	e+02	70.39%
	GOTO	30000	168	70.65	e+01	44.60%
	OPEN	30000	168	128.60	e+02	81.19%
	MKL	30000	168	111.70	e+02	70.52%
<b>C</b>	GOTO	10000	88	66.74	e+01	42.13%
	OPEN	10000	88	117.10	e+02	73.93%
	MKL	10000	108	116.80	e+02	73.74%
	GOTO	20000	108	70.46	e+01	44.48%
	OPEN	20000	128	127.70	e+02	80.62%
	MKL	20000	168	129.10	e+02	81.50%
	GOTO	30000	168	72.40	e+01	45.71%
	OPEN	30000	168	133.00	e+02	83.96%
	MKL	30000	168	136.50	e+02	86.17%
<b>D</b>	GOTO	10000	108	67.24	e+01	42.45%
	OPEN	10000	88	118.60	e+02	74.87%
	MKL	10000	108	119.00	e+02	75.13%
	GOTO	20000	88	71.09	e+01	44.88%
	OPEN	20000	128	128.90	e+02	81.38%
	MKL	20000	168	130.70	e+02	82.51%
	GOTO	30000	168	72.83	e+01	45.98%
	OPEN	30000	168	135.00	e+02	85.23%
	MKL	30000	168	138.20	e+02	87.25%
<b>E</b>	GOTO	10000	108	65.69	e+01	41.47%
	OPEN	10000	88	111.10	e+02	70.14%
	MKL	10000	108	112.00	e+02	70.71%
	GOTO	20000	108	69.32	e+01	43.76%
	OPEN	20000	128	125.60	e+02	79.29%
	MKL	20000	168	118.70	e+02	74.94%
	GOTO	30000	168	71.53	e+01	45.16%
	OPEN	30000	128	130.60	e+02	82.45%
	MKL	30000	168	127.00	e+02	80.18%

FIGURE F.4: GCC 數據統計-i7



### F.3 ICC 統計表

E3-ICC						
架構	函式庫	網格	分區分塊	浮點數	單位	效能比
<b>A</b>	GOTO	10000	88	47.48	e+01	22.48%
	OPEN	10000	168	126.30	e+02	59.80%
	MKL	10000	128	129.60	e+02	61.36%
	GOTO	20000	128	49.64	e+01	23.50%
	OPEN	20000	128	137.30	e+02	65.01%
	MKL	20000	128	149.90	e+02	70.98%
	GOTO	30000	168	50.72	e+01	24.02%
	OPEN	30000	128	148.60	e+02	70.36%
	MKL	30000	168	163.40	e+02	77.37%
<b>B</b>	GOTO	10000	108	48.28	e+01	22.86%
	OPEN	10000	168	127.40	e+02	60.32%
	MKL	10000	128	134.60	e+02	63.73%
	GOTO	20000	108	50.73	e+01	24.02%
	OPEN	20000	168	148.20	e+02	70.17%
	MKL	20000	128	148.60	e+02	70.36%
	GOTO	30000	168	49.87	e+01	23.61%
	OPEN	30000	168	156.20	e+02	73.96%
	MKL	30000	168	162.60	e+02	76.99%
<b>C</b>	GOTO	10000	88	48.82	e+01	23.12%
	OPEN	10000	168	131.50	e+02	62.26%
	MKL	10000	168	140.70	e+02	66.62%
	GOTO	20000	128	51.07	e+01	24.18%
	OPEN	20000	168	151.20	e+02	71.59%
	MKL	20000	168	161.80	e+02	76.61%
	GOTO	30000	168	52.03	e+01	24.64%
	OPEN	30000	168	160.00	e+02	75.76%
	MKL	30000	168	172.30	e+02	81.58%
<b>D</b>	GOTO	10000	88	48.40	e+01	22.92%
	OPEN	10000	168	131.00	e+02	62.03%
	MKL	10000	168	143.20	e+02	67.80%
	GOTO	20000	108	50.66	e+01	23.99%
	OPEN	20000	168	150.30	e+02	71.16%
	MKL	20000	168	163.70	e+02	77.51%
	GOTO	30000	168	51.67	e+01	24.46%
	OPEN	30000	168	159.90	e+02	75.71%
	MKL	30000	168	172.40	e+02	81.63%

FIGURE F.5: ICC 數據統計-E3

i7-ICC						
架構	函式庫	網格	分區分塊	浮點數	單位	效能比
<b>A</b>	GOTO	10000	108	62.42	e+01	39.41%
	OPEN	10000	108	106.30	e+02	67.11%
	MKL	10000	168	99.60	e+01	62.88%
	GOTO	20000	108	66.79	e+01	42.17%
	OPEN	20000	128	119.10	e+02	75.19%
	MKL	20000	168	108.90	e+02	68.75%
	GOTO	30000	168	68.68	e+01	43.36%
	OPEN	30000	168	124.90	e+02	78.85%
	MKL	30000	168	111.60	e+02	70.45%
<b>B</b>	GOTO	10000	108	64.79	e+01	40.90%
	OPEN	10000	88	110.80	e+02	69.95%
	MKL	10000	128	101.20	e+02	63.89%
	GOTO	20000	108	68.80	e+01	43.43%
	OPEN	20000	128	120.30	e+02	75.95%
	MKL	20000	168	111.50	e+02	70.39%
	GOTO	30000	168	71.02	e+01	44.84%
	OPEN	30000	168	129.00	e+02	81.44%
	MKL	30000	168	114.60	e+02	72.35%
<b>C</b>	GOTO	10000	108	66.46	e+01	41.96%
	OPEN	10000	88	115.30	e+02	72.79%
	MKL	10000	108	116.20	e+02	73.36%
	GOTO	20000	108	70.27	e+01	44.36%
	OPEN	20000	128	127.40	e+02	80.43%
	MKL	20000	168	127.40	e+02	80.43%
	GOTO	30000	168	72.32	e+01	45.66%
	OPEN	30000	168	133.50	e+02	84.28%
	MKL	30000	168	135.50	e+02	85.54%
<b>D</b>	GOTO	10000	108	67.46	e+01	42.59%
	OPEN	10000	108	117.30	e+02	74.05%
	MKL	10000	108	121.10	e+02	76.45%
	GOTO	20000	88	71.14	e+01	44.91%
	OPEN	20000	128	129.10	e+02	81.50%
	MKL	20000	168	130.90	e+02	82.64%
	GOTO	30000	168	72.91	e+01	46.03%
	OPEN	30000	168	135.30	e+02	85.42%
	MKL	30000	168	138.30	e+02	87.31%
<b>E</b>	GOTO	10000	108	65.49	e+01	41.34%
	OPEN	10000	168	112.70	e+02	71.15%
	MKL	10000	108	114.60	e+02	72.35%
	GOTO	20000	168	68.87	e+01	43.48%
	OPEN	20000	128	126.40	e+02	79.80%
	MKL	20000	168	127.00	e+02	80.18%
	GOTO	30000	168	71.50	e+01	45.14%
	OPEN	30000	168	132.20	e+02	83.46%
	MKL	30000	168	135.80	e+02	85.73%

FIGURE F.6: ICC 數據統計-i7

## F.4 MKL 統計表

E3-MKL						
架構	函式庫	網格	分區分塊	浮點數	單位	效能比
<b>A</b>	GCC	10000	128	134.60	e+02	63.73%
	ICC	10000	128	129.60	e+02	61.36%
	GCC	20000	128	149.50	e+02	70.79%
	ICC	20000	128	149.90	e+02	70.98%
	GCC	30000	168	162.60	e+02	76.99%
	ICC	30000	168	163.40	e+02	77.37%
<b>B</b>	GCC	10000	128	137.40	e+02	65.06%
	ICC	10000	128	134.60	e+02	63.73%
	GCC	20000	128	153.40	e+02	72.63%
	ICC	20000	128	148.60	e+02	70.36%
	GCC	30000	168	169.20	e+02	80.11%
	ICC	30000	168	162.60	e+02	76.99%
<b>C</b>	GCC	10000	168	144.60	e+02	68.47%
	ICC	10000	168	140.70	e+02	66.62%
	GCC	20000	168	163.10	e+02	77.23%
	ICC	20000	168	161.80	e+02	76.61%
	GCC	30000	168	171.50	e+02	81.20%
	ICC	30000	168	172.30	e+02	81.58%
<b>D</b>	GCC	10000	168	142.90	e+02	67.66%
	ICC	10000	168	143.20	e+02	67.80%
	GCC	20000	168	162.90	e+02	77.13%
	ICC	20000	168	163.70	e+02	77.51%
	GCC	30000	168	172.30	e+02	81.58%
	ICC	30000	168	172.40	e+02	81.63%

FIGURE F.7: MKL 數據統計-E3

i7-MKL						
架構	函式庫	網格	分區分塊	浮點數	單位	效能比
<b>A</b>	GCC	10000	168	100.40	e+02	63.38%
	ICC	10000	168	99.60	e+01	62.88%
	GCC	20000	168	108.80	e+02	68.69%
	ICC	20000	168	108.90	e+02	68.75%
	GCC	30000	168	111.20	e+02	70.20%
	ICC	30000	168	111.60	e+02	70.45%
<b>B</b>	GCC	10000	168	102.20	e+02	64.52%
	ICC	10000	128	101.20	e+02	63.89%
	GCC	20000	168	111.50	e+02	70.39%
	ICC	20000	168	111.50	e+02	70.39%
	GCC	30000	168	111.70	e+02	70.52%
	ICC	30000	168	114.60	e+02	72.35%
<b>C</b>	GCC	10000	108	116.80	e+02	73.74%
	ICC	10000	108	116.20	e+02	73.36%
	GCC	20000	168	129.10	e+02	81.50%
	ICC	20000	168	127.40	e+02	80.43%
	GCC	30000	168	136.50	e+02	86.17%
	ICC	30000	168	135.50	e+02	85.54%
<b>D</b>	GCC	10000	108	119.00	e+02	75.13%
	ICC	10000	108	121.10	e+02	76.45%
	GCC	20000	168	130.70	e+02	82.51%
	ICC	20000	168	130.90	e+02	82.64%
	GCC	30000	168	138.20	e+02	87.25%
	ICC	30000	168	138.30	e+02	87.31%
<b>E</b>	GCC	10000	108	112.00	e+02	70.71%
	ICC	10000	108	114.60	e+02	72.35%
	GCC	20000	168	118.70	e+02	74.94%
	ICC	20000	168	127.00	e+02	80.18%
	GCC	30000	168	127.00	e+02	80.18%
	ICC	30000	168	135.80	e+02	85.73%

FIGURE F.8: MKL 數據統計-i7

## F.5 分區分塊因子統計表

Num	Tool+Blas	Grid	Nb
A	HPL_GOTO	10000	108
	HPL_OPEN	10000	128
B	HPL_GOTO	10000	108
	HPL_OPEN	10000	128
C	HPL_GOTO	10000	108
	HPL_OPEN	10000	128
D	HPL_GOTO	10000	88
	HPL_OPEN	10000	128

E3統計分析	
分區分塊因子之精細度落點	
Nb	Number
88	4
108	11
128	32
168	33
<b>總數</b>	<b>80</b>

Num	Tool+Blas	Grid	Nb	Num	Tool+Blas	Grid	Nb
A	GCC_GOTO	10000	108	A	ICC_GOTO	10000	88
	GCC_OPEN	10000	128		ICC_OPEN	10000	168
	GCC_MKL	10000	128		ICC_MKL	10000	128
	GCC_GOTO	20000	128		ICC_GOTO	20000	128
	GCC_OPEN	20000	128		ICC_OPEN	20000	128
	GCC_MKL	20000	128		ICC_MKL	20000	128
	GCC_GOTO	30000	128		ICC_GOTO	30000	168
	GCC_OPEN	30000	128		ICC_OPEN	30000	128
	GCC_MKL	30000	168		ICC_MKL	30000	168
B	GCC_GOTO	10000	108	B	ICC_GOTO	10000	108
	GCC_OPEN	10000	128		ICC_OPEN	10000	168
	GCC_MKL	10000	128		ICC_MKL	10000	128
	GCC_GOTO	20000	128		ICC_GOTO	20000	108
	GCC_OPEN	20000	128		ICC_OPEN	20000	168
	GCC_MKL	20000	128		ICC_MKL	20000	128
	GCC_GOTO	30000	168		ICC_GOTO	30000	168
	GCC_OPEN	30000	128		ICC_OPEN	30000	168
	GCC_MKL	30000	168		ICC_MKL	30000	168
C	GCC_GOTO	10000	108	C	ICC_GOTO	10000	88
	GCC_OPEN	10000	128		ICC_OPEN	10000	168
	GCC_MKL	10000	168		ICC_MKL	10000	168
	GCC_GOTO	20000	108		ICC_GOTO	20000	128
	GCC_OPEN	20000	128		ICC_OPEN	20000	168
	GCC_MKL	20000	168		ICC_MKL	20000	168
	GCC_GOTO	30000	128		ICC_GOTO	30000	168
	GCC_OPEN	30000	128		ICC_OPEN	30000	168
	GCC_MKL	30000	168		ICC_MKL	30000	168
D	GCC_GOTO	10000	108	D	ICC_GOTO	10000	88
	GCC_OPEN	10000	128		ICC_OPEN	10000	168
	GCC_MKL	10000	168		ICC_MKL	10000	168
	GCC_GOTO	20000	168		ICC_GOTO	20000	108
	GCC_OPEN	20000	128		ICC_OPEN	20000	168
	GCC_MKL	20000	168		ICC_MKL	20000	168
	GCC_GOTO	30000	168		ICC_GOTO	30000	168
	GCC_OPEN	30000	128		ICC_OPEN	30000	168
	GCC_MKL	30000	168		ICC_MKL	30000	168

HPL分析	
Nb	Number
88	1
108	3
128	4
168	0
<b>小計</b>	<b>8</b>

GCC分析	
Nb	Number
88	0
108	5
128	20
168	11
<b>小計</b>	<b>36</b>

ICC分析	
Nb	Number
88	3
108	3
128	8
168	22
<b>小計</b>	<b>36</b>

FIGURE F.9: 分區分塊因子數據統計-E3



Num	Tool+Blas	Grid	Nb
A	HPL_GOTO	10000	108
	HPL_OPEN	10000	128
B	HPL_GOTO	10000	108
	HPL_OPEN	10000	88
C	HPL_GOTO	10000	108
	HPL_OPEN	10000	88
D	HPL_GOTO	10000	88
	HPL_OPEN	10000	88

Num	Tool+Blas	Grid	Nb
A	GCC_GOTO	10000	108
	GCC_OPEN	10000	108
	GCC_MKL	10000	168
	GCC_GOTO	20000	108
	GCC_OPEN	20000	128
	GCC_MKL	20000	168
	GCC_GOTO	30000	168
	GCC_OPEN	30000	168
	GCC_MKL	30000	168
B	GCC_GOTO	10000	88
	GCC_OPEN	10000	88
	GCC_MKL	10000	168
	GCC_GOTO	20000	108
	GCC_OPEN	20000	128
	GCC_MKL	20000	168
	GCC_GOTO	30000	168
	GCC_OPEN	30000	168
	GCC_MKL	30000	168
C	GCC_GOTO	10000	88
	GCC_OPEN	10000	88
	GCC_MKL	10000	108
	GCC_GOTO	20000	108
	GCC_OPEN	20000	128
	GCC_MKL	20000	168
	GCC_GOTO	30000	168
	GCC_OPEN	30000	168
	GCC_MKL	30000	168
D	GCC_GOTO	10000	108
	GCC_OPEN	10000	88
	GCC_MKL	10000	108
	GCC_GOTO	20000	88
	GCC_OPEN	20000	128
	GCC_MKL	20000	168
	GCC_GOTO	30000	168
	GCC_OPEN	30000	168
	GCC_MKL	30000	168

i7統計分析	
分區分塊因子之精細度落點	
Nb	Number
88	13
108	22
128	10
168	35
總數	80

HPL分析	
Nb	Number
88	4
108	3
128	1
168	0
小計	8

GCC分析	
Nb	Number
88	6
108	8
128	4
168	18
小計	36

ICC分析	
Nb	Number
88	3
108	11
128	5
168	17
小計	36

FIGURE F.10: 分區分塊因子數據統計-i7

Num	Tool+Blas	Grid	Nb
<b>E</b>	HPL_GOTO	10000	108
	HPL_OPEN	10000	128

i7統計分析-KVM	
分區分塊因子之精細度落點	
Nb	Number
<b>88</b>	<b>0</b>
<b>108</b>	<b>7</b>
<b>128</b>	<b>3</b>
<b>168</b>	<b>10</b>
<b>總數</b>	<b>20</b>

Num	Tool+Blas	Grid	Nb	Num	Tool+Blas	Grid	Nb
<b>E</b>	GCC_GOTO	10000	108	<b>E</b>	ICC_GOTO	10000	108
	GCC_OPEN	10000	108		ICC_OPEN	10000	108
	GCC_MKL	10000	168		ICC_MKL	10000	168
	GCC_GOTO	20000	108		ICC_GOTO	20000	108
	GCC_OPEN	20000	128		ICC_OPEN	20000	128
	GCC_MKL	20000	168		ICC_MKL	20000	168
	GCC_GOTO	30000	168		ICC_GOTO	30000	168
	GCC_OPEN	30000	168		ICC_OPEN	30000	168
	GCC_MKL	30000	168		ICC_MKL	30000	168

HPL分析	
Nb	Number
88	0
108	1
128	1
168	0
<b>小計</b>	<b>2</b>

GCC分析	
Nb	Number
88	0
108	3
128	1
168	5
<b>小計</b>	<b>9</b>

ICC分析	
Nb	Number
88	0
108	3
128	1
168	5
<b>小計</b>	<b>9</b>

FIGURE F.11: 分區分塊因子數據統計-i7-KVM

## 附錄 G

# Intel CPU Gflops

## G.1 Haswell、Sandy Bridge 指令集倍率對照表

### Haswell新计算指令集

- 英特尔®高级矢量扩展指令集2 (Intel® AVX2)

- 包括

- 256-bit整数矢量
- FMA:融合乘加
- 全宽度元素置换
- 聚合

- 优势

- 高性能计算
- 音频和视频处理
- 游戏处理

- 新的整数指令

- 索引和散列指令
- 加密/解密
- 字节序转换 - MOVBE

	指令集	每周期单精度FLOPs	每周期双精度FLOPs
Nehalem	SSE (128-bits)	8	4
Sandy Bridge	AVX (256-bits)	16	8
Haswell	AVX2 & FMA	32	16

类别	指令
比特封装/解析	BZHI, SHLX, SHRX, SARX, BEXTR
变量比特长度流解码	LZCNT, TZCNT, BLSR, BLSMSK, BLSI, ANDN
比特聚合/分散	PDEP, PEXT
随机精度算法 & 散列	MULX, RORX

- 在此获取完整的指令说明书: <http://software.intel.com/en-us/avx/>

Intel® AVX 2 & BMI的专门课程:  
ARCS003 - 周三 15:45 在 307A 会议室

IDF2013  
英特尔信息技术峰会

13 Intel® Microarchitecture (Haswell); Intel® Microarchitecture (Sandy Bridge); Intel® Microarchitecture (Nehalem)

FIGURE G.1: Haswell、Sandy Bridge 指令集倍率對照表



## G.2 E3-1230 V3

RE: Intel@Xeon® E3-1230 V3 - 郵件 (HTML)

2016/6/8 (週二) 上午 07:05

### CTP Determinations

RE: Intel@Xeon® E3-1230 V3

收件者: g03357006  
副本: Gentry, Eric

您已於 2016/6/8 上午 11:43 回覆此訊息。

Hello,

Processor#	Base clock GHz	APP value(WT)			CTP Value (MTOps)		GFLOPS		Single-Core Max Turbo Boost Clock GHz	Single-Core Max Turbo APP value(WT)			Single-Core Max Turbo CTP Value (MTOps)		Single-Core Max Turbo GFLOPS	
		1way	2way	4way	CPU 1-way	GPU 1-way	CPU 1-way	GPU 1-way		1way	2way	4way	CPU 1-way	GPU 1-way	CPU 1-way	GPU 1-way
E3-1230v3	3.3	0.063360	0.126720	0.253440	267,300	N/A	211.2	N/A	3.7	0.071040	0.142080	0.284160	299,700	N/A	236.8	N/A

APP values are based on single core max turbo frequency, which is consistent with the information published on [www.intel.com](http://www.intel.com). Effective immediately, Intel will only be providing our customers with the active single core maximum Turbo Boost Technology frequency APP information.

All GFLOPS, CTP and APP calculations contained herein were based on specifications taken from Intel datasheets and are subject to change without notice. Intel makes no representation or warranty as to the accuracy or reliability of such specifications. THESE CALCULATIONS ARE PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in these calculations. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Regards,  
CTP Determinations

**From:** g03357006  
**Sent:** Wednesday, June 8, 2016 2:10 AM  
**To:** CTP Determinations  
**Subject:** Intel@Xeon® E3-1230 V3

CTP Determinations: 沒有項目

FIGURE G.2: Intel CPU E3-1230 V3 Gflops

ARK 功能表 ▶

在此輸入以搜尋產品



## 產品 (原名 Haswell)

### 描述:

Haswell is the codename for the Intel processor microarchitecture that is the successor to the Ivy Bridge microarchitecture. Most Haswell products are branded as 4<sup>th</sup> Generation Intel® Core™ Processors for client systems, and Intel® Xeon® v3 Processors for server systems, in addition to some Pentium and Celeron-branded processors. Haswell is built on the 22-nm manufacturing process (lithography). Intel officially announced processors with this microarchitecture in 2013. Haswell delivers significant performance advancements over previous architectures, including improved graphics, battery life, and security.

全部 (256)	筆記型 (79)	桌上型電腦 (72)	伺服器 (72)	嵌入式 (52)			
比較	產品名稱	狀態	推出日期	核心數量	TDP	建議客戶價格	處理器繪圖#
比較全部 +							
比較 +	Intel® Xeon® Processor E3-1230 v3 (8M Cache, 3.30 GHz)	End of Life	Q2'13	4	80 W	\$240.00 - \$250.00	None
比較 +	Intel® Xeon® Processor E3-1230L v3 (8M Cache, 1.80 GHz)	Launched	Q2'13	4	25 W	\$250.00	None
比較 +	Intel® Xeon® Processor E3-1231 v3 (8M Cache, 3.40 GHz)	Launched	Q2'14	4	80 W	\$240.00 - \$250.00	None
比較 +	Intel® Xeon® Processor E3-1240 v3 (8M Cache, 3.40 GHz)	End of Life	Q2'13	4	80 W	\$262.00 - \$273.00	None
比較 +	Intel® Xeon® Processor E3-1240L v3 (8M Cache, 2.00 GHz)	Launched	Q2'14	4	25 W	\$278.00	None

FIGURE G.3: E3-1230 V3 Haswell 確認

## G.3 i7-3960X



### Intel® Core i7-3900 Desktop Processor Extreme Edition Series

Processor Number	Frequency Type	Clock GHz	CTP	GFLOP	APP 1-way	APP 2-way	APP 4-way
i7-3960X	Base	3.3	195800	158.4	0.04752	0.09504	0.19008
	Single Core Max Turbo	3.9	231400	187	0.05616	0.11232	0.22464
	GPU ONLY	N/A	N/A	N/A	N/A	N/A	N/A

Intel Corporation is providing the Product Export Compliance Matrix which contains the information necessary to complete an export assessment against the US Export Administration Regulations and local export country regulations such as Adjusted Peak Performance (APP) in Weighted Teraflops (WT), Export Control Classification Number (ECCN), AND/OR Harmonized Tariff Number (HTS), etc. APP values are calculated using the Single Core Max Turbo Frequency as published on ark.intel.com. Effective immediately, Intel will only be providing our customers with the APP information for the single core and multi-core processors based on the Single Core Maximum Turbo Boost Technology Frequency information, if applicable.

All information provided in the Product Export Compliance Matrix is strictly a recommendation to the user, and should be used in conjunction with the US Export Administration Regulations (EAR) and local country export regulations when assessing export requirements for the US and local export country.

All the information contained herein are based on Intel product specifications and are subject to change without notice. Intel makes no representation or warranty as to the accuracy or reliability of such specifications. These calculations are provided "AS IS" with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose or any warranty otherwise arising out of any proposal, specification or sample. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in these calculations. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

FIGURE G.4: Intel CPU i7-3960X Gflops