Beowulf 中文HOWTO

Jacek Radajewski and Douglas Eadline

v1.1.1, 22 November 1998

這份文件介紹Beowulf超級電腦架構和對撰寫平行程式提供一些背景知識，並包括一些有趣的文件和網站連接。

目錄

1 前言

1.1 聲明

我們將不對本文件中不正確的資訊，或者當使用後造成的損失負責。

1.2 版權

1.3 有關這份HOWTO

Jacek Radajewski在1997年十一月開始編寫這份文件，很快地Douglas Eadline也加入行列，幾個月後Beowulf HOWTO變成長篇的文件，並在1998年八月分成三份，Beowulf HOWTO、Beowulf 架構設計HOWTO 和Beowulf安裝管理 HOWTO。1998年十一月十一日Beowulf HOWTO 1.0.0版本在Linux 文件計畫上發佈，我們希望這只是完整的Beowulf 文件計畫的第一步。

1.4 有關這些作者

- Jacek Radajewski 網管人員，現正在澳洲南昆士蘭大學攻讀電腦科學。 Jacek於1995年第一次接觸Linux就深受吸引，他在1997年建立他自己的第一套 Beowulf電腦群，從那時起開始研究，經常想找到更新和更好的方法架設系統。 妳可以透過電子郵件與他聯繫*jacek@usq.edu.au*。

- Douglas Eadline 博士，也是美國Paralogic公司的董事長和主要科學家。 原主修物理及分析化學，1978年建立個人第一台電腦用在化學儀器上， 這才開始進入電腦的領域，Eadline博士現在興趣包括Linux，Beowulf電腦群和 平行algorithms，妳可以透過電子郵件與他聯繫*deadline@plogic.com*

1.5 答謝

經過長時期的撰寫，最後終於完成，特別感謝以下朋友對本文件的幫忙和貢獻。

- Becky 的愛、支持和諒解。
- Tom Sterling, Don Becker, 和在NASA 參與Beowulf計畫的朋友。
- Thanh Tran-Cong 和架設*topcat* Beowulf 機器的機械量測的教職員。
- 提供許多建議的指導老師Christopher Vance。
- 我的朋友 Russell Waldron 提供程式方面的點子和他熱心的參與及支持。
- 我的朋友 David Smith 的校稿。
- 許多在Beowulf mailing list上的朋友提供的反應及建議。
- 所有負責Linux作業系統和在*topcat*及其他Beowulf機器上使用軟體套件的朋友。

2 簡介

當容易取得的電腦和網路硬體的效能提升，並且價格不斷滑落，利用這些市面販售的電腦組裝成平行電腦就變得非常可行，而不需要花錢在價格高昂的超級電腦上。事實上Beowulf電腦的價格效能比是傳統超級電腦的三倍到十倍，Beowulf架構在scale上也不錯，它很容易建置，你只需負擔硬體設備的費用，而不須負擔軟體的費用。

2.1 誰需要閱讀這份HOWTO？

這份HOWTO是設計給對Linux作業系統有些認識的人，對於Beowulf技術、作業系統有深入的認識和網路概念的了解都不是必需的，但是對平行計算有些經驗是絕對有好處的（畢竟你總需要些藉口閱讀這份文件）。這份HOWTO文件無法回答Beowulf相關的所有問題，但是希望能提供一些想法，並且領導你走向正確的方向，這份文件的目的是提供相關的背景知識，和一些更深入的參考文件。

2.2 什麼是Beowulf？

Famed was this Beowulf: far flew the boast of him, son of Scyld, in the Scandian lands. So becomes it a youth to quit him well with his father's friends, by fee and gift, that to aid him, aged, in after days, come warriors willing, should war draw nigh, liegemen loyal: by lauded deeds shall an earl have honor in every clan. Beowulf是用英文書寫的最早史詩，描述一個擁有神力和勇氣的英雄人物—北歐武夫，和他力戰怪獸Grendel的故事， 參見5.7 (History) 可以找到更多有關北歐武夫的事蹟。

Beowulf的定義可能和建造者或Beowulf超級電腦使用者一樣多，有些宣稱唯有和NASA原型機相同才稱爲Beowulf，也有人認爲只要在一群工作站上執行平行程式就可以稱爲Beowulf。我對Beowulf的定義是介於二者之間，主要根據Beowulf mailing list上的一些張貼郵件。

Beowulf是一種用來作平行計算的電腦群架構，通常是由一台伺服端和一台以上用户端透過乙太網路或其他網路連接的系統，它是用市面販售的硬體（像是裝有Linux的個人電腦）、標準乙太網路卡和交換式集線器，它不包含任何特殊的硬體設備，是可以重新製造。Beowulf並且使用容易取得的軟體，如Linux作業系統、PVM(Parallel Virtual Machine)和MPI(Message Passing Interface）。伺服端控制整個電腦群，提供用户端檔案服務，它並且也是電腦群的控制台和通訊閘，提供對外連接的出口。大型的Beowulf電腦群可能不只一台伺服端，可能有些電腦有特定用途，例如控制台或監視站，用户端在大多數情況下是不做額外的事情，額外事情做的越少越好。用户端是由伺服端規劃和控制，只做它們被分派的工作。在一個無硬碟用户端的架構，用户端甚至不必知道各自的IP位址或名稱，一直到伺服端告知爲止。Beowulf和工作站群最大的差異點在於Beowulf看起來比較像一台機器，而不是許多台工作站。在大部分情形用户端不需要備有鍵盤和螢幕，只需要透過遠端登入或序列式終端螢幕登入，Beowulf的用户端可視爲處理器和記憶體的組合，嵌入一個電腦群內，就像處理器或記憶體模組被嵌入主機板上。

Beowulf不屬於特定軟體套件、嶄新的網路拓撲或是最新的核心駭客，Beowulf是一套串連Linux電腦成超級虛擬平行電腦的技術，雖然有許多的相關軟體套件，如修改過的核心、PVM和MPI程式庫以及規劃工具（可以讓Beowulf架構更快速、便捷地規劃系統），任何人也都可以只靠Linux標準套件，不需要額外的輔助就可以建立一套標準的Beowulf機器。假如你有二台已經上網的Linux電腦，並且可以透過NFS共用`/home` 檔案系統和執行rsh（remote shells）指令，這樣你可以算是擁有一台簡單的雙節點Beowulf機器。

2.3 分類

Beowulf系統已經由各種不同部份組成，爲了效能的考量，一些非商品化周邊設備（只由一家製造商生產）已經問世，爲了易於解說和討論各種不同類型的系統，我們提出下列簡單的分類方式:

第一類BEOWULF:

這類機器完全由商品化、直接從市面上販售的零件所組成，我們用電腦購買者（Computer Shopper）認定標準來定義商品化、直接從市面上販售的零件（電腦購買者是一種每月出版的雜誌，有一英吋厚，內容介紹各種電腦系統和零件目錄），認定標準如下:

A 第一類Beowulf是一種機器，它的組成零件至少必須在三種國際性或全球性商業目錄上找到。

第一類系統的好處有:

- 硬體設備有很多來源（價格低、容易維護）
- 不會依賴特定一家硬體供應商
- Linux支援的驅動程式

- 通常有一標準基礎（如SCSI、Ethernet等）

第一類系統的缺點有:

- 需要第二類系統的硬體才有較好的效能

第二類BEOWULF

任何沒有通過電腦購買者認定標準的機器稱之。這並不一定是件壞事，它只是分類的一種。

第二類系統的好處有:

- 效能相當地好

第二類系統的缺點有：

- 驅動程式的供應經常更換
- 依賴特定一家硬體供應商
- 可能比第一類系統昂貴

沒有一種類別比其他的優秀，全憑使用者的需求和預算，這種分類純粹希望讓接下來的討論更加簡明，後頭的系統設計章節將會幫助你決定哪一種系統最符合你的要求。

3 架構簡介

3.1 它長什麼樣？

我認爲描述Beowulf超級電腦架構最合適的方法是舉一個眞實的範例，並且是大多數系統管理者所熟悉的。那就是一個UNIX主機實驗室，內有一台伺服端和一群用户端，更精準地說，我會舉位在南昆士蘭大學理學院DEC Alpha大學部計算機實驗室爲例，伺服器被稱爲 *beldin* 用户端機器分別稱爲 *scilab01*, *scilab02*, *scilab03*, 一直到 *scilab20*. 每台用户端內部都安裝Digital Unix 4.0作業系統，但是使用者檔案空間（`/home`）和 `/usr/local` 都是透過NFS（網路檔案系統）從伺服端上獲得，每個用户端都可以進入伺服端，並且所有其他的用户端都會記載在 `/etc/hosts.equiv` 檔案內，因此每個用户端都可以用遠端操作殼（rsh）。伺服端也是整個實驗室的NIS伺服器，因此所有的機器都有相同的帳號資料，某人可以坐在*scilab02*的控制台前登入，就像他登入伺服端或*scilab15*. 一樣有相同的環境，所有的用户端有相同環境的原因在於所有的機器都安裝和規劃相同的作業系統，並且使用者的`/home` 和 `/usr/local` 區域實體上都位在伺服端上，可以透過NFS進入。NIS和NFS更進一步的 訊息請參閱 *NIS* 和 *NFS* HOWTOs.

3.2 如何有效利用其他節點？

現在我們對系統架構有些概念，讓我們看看如何使用計算機實驗室內可供使用的CPU。任何人可以登入任何一台機器，並且在每個人自己的目錄下執行程式，他們也可以透過遠端操作殼在其他電腦上啓動（spawn）相同的程式。舉例來說，假設我們要計算1到10內整數平方根的總和，我們寫了個簡單的程式名爲 `sigmasqrt` （請參見6.2 (source code)），爲了得到結果，我們執行以下的步驟

```
[jacek@beldin sigmasqrt]$ time ./sigmasqrt 1 10
22.468278

real    0m0.029s
user    0m0.001s
sys     0m0.024s
```

time 指令可以告訴我們執行程式所花的時間（實際經過的時間），我們可以看到，這個例子只花了很短的時間（0.029秒），假如我想計算1到1,000,000,000內整數的平方根總和，讓我們試試看，重新計算所花的時間

```
[jacek@beldin sigmasqrt]$ time ./sigmasqrt 1 1000000000
21081851083600.559000

real    16m45.937s
user    16m43.527s
sys     0m0.108s
```

這次執行程式所花的時間非常久，一個明顯的問題就是我們如何加快執行的時間？我們該如何改變執行程式的方式以減少執行所花的時間？最明顯的答案就是將整個工作分成許多小工作，並且同時在所有的電腦上執行，我們可以將加法的工作分成二十份，每個部份做一段開根號的工作，並加起來，當所有的節點完成計算，並傳回來，將二十個數加起來就得到最後的答案。在執行程式之前，我們需要做個標有記號的輸送管，可以讓所有的行程寫下它們的結果。

```
[jacek@beldin sigmasqrt]$ mkfifo output
[jacek@beldin sigmasqrt]$ ./prun.sh & time cat output | ./sum
[1] 5085
21081851083600.941000
[1]+  Done                    ./prun.sh

real    0m58.539s
user    0m0.061s
sys     0m0.206s
```

這回我們花了大約58.5秒，這時間是從開始到所有的節點都完成計算，並將結果寫到輸送管，這個時間並不包括最後將二十個數加起來，不過那個時間非常地短，可以忽略不計。我們可以看到平行計算可以有效地改進執行程式，事實上這個平行工作整整快了約17倍，相對於使用了二十倍CPU數目，效能是相當合理的。上述範例的目的是要展示同時平行程式最簡單的方法，實際操作上，如此簡單的範例是很少見的，其他技巧（PVM和MPI APIs）經常用來達成平行的工作。

3.3 Beowulf如何與COW不同？

上述的計算機實驗室算是一個工作站群（Cluster of Workstations，COW），那麼Beowulf有何不同？它和COW有何差異？實際上二者沒多大差別，不過Beowulf倒是有些不同的特色。第一、大多數的Beowulf群的用户端沒有鍵盤、滑鼠、顯示卡和螢幕，所有到用户端的方式都是從伺服端、特定控制端或是序列控制端經過遠端連接登入，因爲對用户端而言，從電腦群外登入電腦或是從外頭的電腦直接登

入用户端是沒有必要的，用户端通常是使用私有的IP位址，例如從10.0.0.0到10.0.0.8或是192.168.0.0到192.168.0.16（參見RFC 1918 *http://www.alternic.net/rfcs/1900/rfc1918.txt.html*). 通常唯一要用到第二張網路卡對外連接的機器是伺服端，使用這套電腦群最常見的方法是直接進入伺服端，或是從個人工作站使用telnet或遠端登入伺服端。一但進入伺服端，使用者可以編輯和編譯他們的程式，也可以在電腦群內的用户端上啓動行程。大多數情形的COW是在晚上用來執行平行計算，和在人們不使用工作站的週末時間，使用閒置的CPU。而通常Beowulf專用來平行計算，並且對這些平行計算做最佳化，當利用市售電腦零組件和免費軟體建構的Beowulf也提供較好的價格效能比，並且Beowulf給人一種單一系統的印象，很容易讓使用者將Beowulf群看作是一台計算用工作站。

4 系統設計

在購買任何硬體設備之前，思考如何設計你要的系統是非常重要的，基本上在設計一套Beowulf系統有兩項硬體設備的議題：你將使用的節點或稱電腦的機型和連接節點的方式，只有一種軟體議題會影響你要選擇的硬體設備，就是通訊用程式庫或稱API，更詳細的硬體設備和通訊軟體討論將會在本文件後頭。

當選擇性不多的時候，有幾樣重要的設計決定必須做的，因爲平行計算的科學（或稱爲藝術）有很多種方式解釋，稍後有作簡介，假如你不想看一些背景資料，可以跳過本節，但是建議你在做硬體設備最後的決定之前，最好先閱讀 4.4 (可適性（Suitability）)。

4.1 平行計算的背景介紹

本節提供一些平行計算觀念的基本知識，這絕不是平行計算科學和技術的詳細描述，這只是平行計算中與Beowulf設計者和使用者相關的一些簡介。

當你要設計和建構自己的Beowulf，下列即將描述的許多項議題在你做決定的過程中將會變得非常重要，肇因於它的零件特性，一個Beowulf超級電腦可被我們所掌控，一些因素就得仔細考量，一般來說，平行計算所牽扯的議題並不太難了解，的確如此，一旦了解這些議題，個人的期待將會實踐，成功將更容易實現。不像循序的世界，處理器的速度是唯一最重要的因素，在平行的世界中，處理器速度只是決定整個系統效能和效益數個因素之一。

4.2 平行計算的方法

平行計算可分成好幾種類型，從使用者觀點，考慮每種類型的優缺點都很重要，接下來的章節嘗試提供平行計算方法的觀點，並指出Beowulf機器是屬於哪種。

4.2.1 爲什麼要一顆以上的處理器?

回答這個問題是很重要的，用八顆CPU跑文書處理軟體聽起來似乎有點殺雞用牛刀—實際上也是這樣。那其他像是web server、database、rendering program或是project scheduler？額外的CPU可能有所幫助。複雜的數值模擬又如何？流體動力學碼或是data mining application，在這些狀況下，額外的CPU絕對是需要的，的確，多處理器可以用來解決更多問題。

接下來的問題通常是「爲什麼我需要二或四顆CPU？我可以等極速的986出現。」，有下列幾個原因可以回答這個問題:

1. 由於使用多工作業系統，電腦可以同時做很多事情，只要有一顆以上低價CPU就可以達到最自然的平行化。
2. 處理器的速度每十八個月就增加一倍，但是記憶體和硬碟的速度呢？很不幸地，它們的速度增加地並沒有CPU快，我們必須記得，大多數的應用軟體都利用到cache以外的記憶體和硬碟，平行化是可以擺脫這些限制的一種方法。
3. 預計在2005年之後，處理器的速度將不會再每十八個月就增加一倍，要達到這種曾加速度的趨勢，中間還有許多障礙。
4. 平行計算可以有2倍到500倍速度的提升（有時更快），這全看執行的應用程式。這種提升是無法單靠單一處理器，即使是曾經一度使用特別設計超快處理器的超級電腦，如今也是用多顆市售且隨手取得的CPU所組成。

假如你因爲遇到計算限制（computer bound）或是輸出／輸入限制（I/O bound）而需要速度，平行是值得考慮的方法，因爲平行計算可以有很多方式，平行地解決你的問題將要思考許多重要的抉擇，這些抉擇會嚴重影響應用程式的可攜性、效能和你所花的時間、精神以及金錢。

在我們了解平行計算的技術之前，讓我們先看看熟悉的眞正平行計算問題的實例—在店門前大排長龍。

4.2.2 平行計算的商店

想想看一個櫃台前有八台同時使用收銀機的大型商店，假設每個收銀機就是一顆CPU，每個客人就是一個電腦程式，電腦程式的大小（工作的多寡）就是每個客人選點的多少，接下來所作類比的方式就是要說明平行計算的觀念。

單工作業系統　有一台收銀台打開，一次只能處理一個客人。

範例：MS DOS

多工作業系統　有一台收銀機打開，但是現在一次處理一個客人選點的一部份，然後移到下個客人，處理他選點的一部份，每個人似乎同時都有在移動，假如沒有其他人在排隊，很快就會輪到你。

範例：UNIX，使用單一CPU的NT。

多顆CPU且多工作業系統　在我們將其他的收銀機打開，每個客人都有一台收銀機服務，這時排隊的隊伍移動地很快，這稱爲SMP—對稱式多行程。雖然有額外的收銀機打開，但是絕快不過只有一台收銀機和一個客人。

範例：UNIX，使用多CPU的NT

多工多CPU的緒（thread）　如將你選點的項目拆開來，多台收銀機同時使用記帳，你就可以更快一點。首先我們得假設你買了很多東西，因爲你花在拆開項目的時間必須由多台收銀機補償回來，理論上，你可以比以前快N倍，N是收銀機的台數。當收銀員需要得到其他部份的小計時，他們可以透過交談或觀望其他收銀機，很快地交換他們所需要的資訊，他們甚至可以打探其他的收銀機，找尋需要的資料，使得工作更快些。無論如何都還是有些限制，也就是這家店在各個地方可以有效地放置多少台收銀機。

Amdals定律也使應用程式增快的速度將受限在循序程式中最慢的部份。

範例：NUIX或是在相同主機板上的多CPU的NT並可以執行多緒（multi-threaded）程式。

在多工作業系統上向其他CPU傳遞訊息 了改善效能，店家在後頭又增加了八台收銀機，因爲新的帳單離前方櫃台很遠，收銀員必須用電話將小計告訴前方櫃台，除了傳遞外，還加上額外時間的負擔，但是假如傳遞時間很短，它將不會造成問題。假如妳要買的東西很多，需要所有的收銀機，這時在使用所有收銀機來改進收帳的速度之前，額外的時間負擔仍須考慮進去。有時候，某些商店在各個角落只單獨放置一台收銀機，每個收銀機就只能透過電話聯繫，這時它們所在的位置就不重要了。

範例：多台UNIX或多CPU的NT，可能在同一張主機板或許多主機板上，彼此能相互聯繫。

上述說明雖然不夠精準，但對平行系統的限制來說，算是不錯的描述，不像單一CPU的傳遞仍是個議題。

4.3 平行計算的架構

平行計算的方法和架構將在下節介紹，雖然描述將會很廣泛，但是也足以了解Beowulf設計的一些相關議題。

4.3.1 硬體架構

在硬體上有二種基本的平行電腦：

1. 自有記憶體機器，之間可以交換資訊（Beowulf 電腦群）。
2. 共享記憶體機器，透過記憶體傳遞資料（SMP機器）。

典型的Beowulf是由一群單CPU機器組成，透過高速乙太網路連接，所以稱爲自有記憶體機器。4 way SMP是一台共享記憶體機器，可用來作平行計算，平行的應用軟體透過共享記憶體傳遞資料。以電腦販售店做比喻，自有記憶體機器（單獨暫存帳單）在CPU數量上可以很多，但是共享記憶體機器由於記憶體的關係，CPU的數目是有限制的。

但是連接多台共享記憶體機器是可行的，這些混合式共享記憶體機器對使用者看起來就像一台大型的SMP，經常稱作駑馬（NUMA，non uniform memory access，非均勻記憶體登入），因爲使用者看到的是一塊大記憶體，由所有的CPU共享，有著各種不同的延遲（latencies）。在某種程度上，駑馬機器中各個自有共享記憶體之間是必須互相傳遞訊息。

把SMP機器當作自有記憶體的計算節點，並將它們連接起來是有可能的。典型的第一類主機板可以有二顆或四顆CPU，使用這類電腦通常可以降低整體的成本，Linux內部排序決定如何共享這些CPU，在這個階段，使用者無法指定所要執行的工作由哪個CPU負責，但是使用者可以同時執行二個不相干的行程，或是一個有緒的行程（threaded processes），並希望效能比一個CPU的系統好。

4.3.2 軟體API架構

基本上有二種方式可以在程式內表現出同時的特性：

1. 在處理器之間使用訊息傳送。
2. 使用系統的緒

仍有別種方法，但是這二種是最常用的。有一點必須注意，就是同時不需要由底層的硬體所控制，訊息和緒都可以在SMP、駑馬SMP和電腦群上使用，但如上所述，效能和可攜性仍是重要的議題。

訊息　歷史的觀點來看，訊息傳遞的技術反應出早期自有記憶體平行電腦的設計過程，當緒需要資料時，訊息被要求需要拷貝，拷貝訊息的延遲和速度變成訊息傳遞模式的限制因素。訊息傳遞其實相當簡單，一些資料和傳遞的目的地（處理器）。一般常見訊息傳遞的API有 *PVM* 或 *MPI*，訊息傳遞可以在一台SMP機器和電腦群上有效地使用緒和訊息，相對於緒，訊息傳遞在一台SMP上的好處是，未來一旦妳決定要使用電腦群，只需要輕易地增加機器。

緒　業系統緒的發展主要因爲共享記憶體的SMP設計允許程式中同時的部份可以有很快地共享記憶體傳遞和記憶體同步，緒在SMP系統執行地不錯，這是因爲傳遞是透過共享記憶體，由於這個原因，使用者必須將當地的資料從整體的資料中獨立出來，否則程式將不能正確地執行。相對於訊息傳遞，因爲資料是由行程所共享，大量的資料拷貝可以避免，Linux支援POSIX緒，緒的問題在於很難擴展到一台SMP機器以外，這是因爲資料是由CPU所共享，快閃一致性的議題會造成負擔。將緒有效地擴展到多台SMP機器必須仰賴駕馬技術，但是駕馬非常耗時，並且基本的Linux是不支援的。將緒建構在訊息傳遞之上，曾經有人做過 (*(http://syntron.com/ptools/ptools_pg.htm)*)，但是緒和訊息傳遞在一起就變得效果不佳。

以下是和效能有關的資訊

	SMP機器效能	電腦群效能	比例增加程度
訊息	好	佳	佳
緒	佳	不良*	不良*

* 要求昂貴的駕馬技術。

4.3.3 應用軟體架構

爲了在多CPU下平行地跑應用程式，在同時部份必須被分開來，一個標準的單CPU應用軟體不會比它在多處理器下跑的快，有些工具和編譯器可以做這種工作，但是將程式平行化可不是“隨插即用“。這完全和程式有關，有些程式很容易平行化，有些是極度困難，有些情形受限於algorithm的相關性而根本不可能做到平行。

在討論軟體議題之前，先要介紹合適性的觀念。

4.4 合適性（Suitability）

關平行計算的大多數問題都有相同的答案：

全和應用程式本身有關。

在我們進入這個議題之前，有一個非常重要的不同點需要釐清—同時（CONCURRENT）和平行（PARALLEL）之間的差異性，爲了方便討論起見，我們先定義這二個觀念：

程式內同時的部份是指可以單獨個別計算的部份。

程式內平行的部份是指那些可以在同一時間內分別由不同處理器執行的同時部份。

二者相異的地方是非常重要，因爲同時是程式本身的特性，而有效的平行則是機器的特性，理想狀況下，較快的效能肇因於平行執行，平行效能的限制因素在於計算節點之間的傳遞速度和延遲（延遲也會出現在緒SMP應用軟體，主要來自於快閃（cache）的一致性）。大多數通用的平行測試套件都有很

高的平行性，傳遞和延遲都不是瓶頸，這類問題可以稱作“顯而易見的平行“（obviously parallel），其他的應用軟體就沒那麼簡單，平行地執行程式中的同時部份可能會造成程式跑得較慢，抵消掉其他同時部份所得到的效能。簡單說，傳遞所花費的時間必須從儉省的計算時間補償，否則平行執行同時部份會很不經濟。

程式設計者的工作是要決定程式哪些同時的部份應該平行化，哪些則不要。這將會決定應用程式的效能，下面的圖對程式設計者做了些總結。

```
佔應用程
式的百分比

        | *
        | *
        | *
        | *
        |  *
        |  *
        |  *
        |  *
        |    *
        |     *
        |      *
        |        ****
        |            ****
        |                ********************
        +-----------------------------------
                   傳遞時間 / 計算時間
```

在一個理想的平行電腦，傳遞和計算二者相當，任何同時都可以平行化，很不幸地，眞實的平行電腦（包括共享記憶體機器）都像上圖所示。當設計Beowulf時，使用者必須牢記這圖，因爲對一特定平行電腦，平行效能決定於傳遞時間和計算時間之比，應用程式可能可以在各種平行電腦上執行，但是不能保證一定會有較佳的效能。

一般來說，沒有既可攜性又有效能的平行程式。

上圖還有其他的延伸議題，當效能取決於傳遞和計算比，改變比值中的某一項不表示一定可以提高效能。改變處理器的速度，但不改變傳遞的速度，程式可以有沒直覺性的效果。舉例來說，CPU速度提升二倍或三倍，但保持傳遞速度，可能使妳的程式有較好的平行效果，比循序執行更有效，that is, it may now be faster to run the previousloy PARALLEL parts as SEQUENTIAL。更進一步，平行地執行沒有效率的部份，可以使妳的程式無法達到最快的速度，因此，藉由增加更快的處理器，妳可以讓程式慢下來（妳正讓新的CPU不用它最快的速度執行程式）。

升級到更快的CPU可能反而降慢妳的程式速度。

因此，必須知道妳是否可以用平行硬體環境，妳必須對妳的程式在一特定電腦上的可適性有相當的認識，妳必須知道相當多的議題，包括CPU速度、編譯器、訊息傳遞的API、網路等等。請注意，只認識應用程式是不夠的，妳必須指出程式中計算量最重的部份，但是妳不知道這個部份的傳遞花費，對特定系統，它的傳遞所花的時間可能無法讓程式無法有效地平行化。

最後要說一些常發生的錯誤觀念，我們經常說：一個程式被平行化，但是眞實的情形是程式的同時部份才被平行化，從以上的說明，一個程式並沒有平行化，平行化的效益是機器的特性。

4.5 撰寫和移植平行軟體

一旦妳決定需要平行計算，並且想要設計和架設一套Beowulf，根據上述的討論來思考一些和妳的應用程式有關的建議將是個很好的主意。

一般而言，有兩件事妳能夠做的：

1. 直接架設第一類Beowulf，然後想辦法讓妳的應用程式來適應這套系統，或者在Beowulf上直接跑一個現成的平行應用程式（必須注意上述所提的可攜性和效能的議題）。
2. 先思考一下妳將要在妳的Beowulf上跑的應用程式，然後估計何種類型的硬體和軟體是妳所需要的。

兩種情形妳都要考慮效能的議題，一般而言，有三件事妳需要做：

1. 決定妳的程式中的同時部份。
2. 估計平行效能。
3. 描述出程式中的同時部份。

讓我們一一詳述。

4.5.1 決定妳的程式中的同時部份

這個步驟通常是要考慮將妳的程式平行化，如何平行化將在第二個步驟，現在妳要決定資料的關連性。

>從實際操作的角度來看，應用程式可能有二種形態的同時性：計算（數字的計算）和I/O（資料庫）。雖然大部分情形，計算和I/O同時性是相互正交的（orthogonal），但是有些程式是兩者都需要，有些工具程式可以對現有的程式做同時性的分析，這些工具大部分是爲Fortran程式語言設計的，使用Fortran語言有兩種理由：很早以來，大部分的數字計算程式是用Fortran語言寫的，另外Fortran是很容易分析的。假如沒有可利用的工具，這個步驟對現存的應用程式將是非常困難。

4.5.2 估計平行效能

沒有工具程式的幫助，這個步驟將需要不斷地嘗試錯誤，或是根據舊有經驗來猜測。假如妳心目中已經有特定的應用程式，想要決定這個應用程式是CPU限制（計算限制），還是硬碟限制（I/O限制），根據妳的需求，妳的Beowulf可能會有很大的差異。舉例來說，一個計算限制的問題可能需要一些很快的CPU，高速且低延遲的網路，但是一個I/O限制的問題可能需要較慢的CPU和高速乙太網路。

這個建議令大多數人覺得很訝異，一般的想法是處理器越快越好，這想法當然是正確的，但是妳必須要有不受限制的預算經費，實際情形是要在有限的經費得到最高效能的系統，對一個I/O限制的問題，已有現成的規則（稱作Eadline-Dedkov定律）可供利用。

對兩套有相同累積CPU效能指數的平行電腦而言，一個擁有較慢處理器（一個較慢的處理器間的傳輸網路）對I/O主導的應用程式將會有較佳的效能。

要證明這項規則將會超出本文件的範圍，妳若覺得有趣，可以下載這篇論文 *I/O*主導應用程式在平行電腦上的效能考量（*Performance Considerations for I/O-Dominant Applications on Parallel Computers*）(Postscript 格式 109K) *(ftp://www.plogic.com/pub/papers/exs-pap6.ps)*

旦妳已經決定程式中的同時性是何種形態，妳將需要估計一旦平行處理的話，效能將會如何。參見 5.4 (Software) 有對軟體工具的描述。

若沒有這些工具，妳可以透過這個步驟，自行考量，假如每次計算是以分鐘計，資料傳輸則以秒計，那它將是很好的平行對象，但是記住，假如妳將16分鐘的計算時間拆成32份，而每份的資料傳遞需要數秒鐘，那麼事情將變得嚴重。

4.5.3 描述出程式中的同時部份

有幾種方法找出程式中的同時部份：

1. 明確地平行執行
2. 隱含地平行執行

這二者主要的差別在於明確地平行化取決於使用者，隱含地平行化取決於編譯器。

明確的方法 一些基本的方法是要靠使用者專爲平行電腦來修改原始碼，使用者必須使用 *PVM* 或 *MPI*在程式內增加資訊， 或 是使用POSIX緒（無論如何要牢記心中，緒無法在SMP主機板之間移動）。

明確的方法在實行和除錯上最爲困難，使用者通常在標準Fortran 77或 C/C++原始碼中加入函式。MPI程式庫加入一些函式，使得一些標準平行方法容易實行（例如分散和收集函式），另外還可以使用已經被平行化的標準程式庫。無論如何要將可攜性和效能之間的平衡牢記心中。

從歷史上的理由，大多數數值計算的程式是用Fortran語言所寫的，因此在平行計算中，Fortran是受最大的支援（工具、程式庫等）。現在大多數的程式設計者都是用C語言，或是認爲C語言可以執行地更快，而用C語言重新改寫現存的Fortran應用程式。由於C語言最接近通用的機器語言，C語言較快可能是正確的，但是它也有一些重要的缺陷。C語言使用指標（pointer）會讓資料相關性的決定極度困難，自動分析指標也是極度困難，假如妳有現成的Fortran程式，並且未來想要變成平行程式—千萬不要把它轉成C語言。

隱含的方法 含方法是使用者放棄一些或全部放棄自行平行，改用編譯器的一種方法，例如 FORTRAN 90， 高效能Frotran (High Performance Fortran，HPF), 大量協同平行（Bulk Synchronous Parallel，BSP)還有許多正在發展當中。

隱含方法仍要求使用者對於程式同時的特性提供一些資訊，但是編譯器必須對如何平行地執行同時性做出許多決定，這些方法提供某種程度的可攜性和效能，但是對一個平行編譯器，仍然沒有一個最好的方法來描述同時性的問題。

5 Beowulf資源

5.1 起點

- Beowulf mailing list. 只要電子郵件寄到 *beowulf-request@cesdis.gsfc.nasa.gov*，在郵件內容填上 *subscribe*。
- Beowulf 網頁 *http://www.beowulf.org*
- Extreme Linux *http://www.extremelinux.org*
- 在RedHat網站上的Extreme Linux 軟體 *http://www.redhat.com/extreme*

5.2 文件

- Beowulf HOWTO最新版本 *http://www.sci.usq.edu.au/staff/jacek/beowulf.*
- 架設一個Beowulf系統 *http://www.cacr.caltech.edu/beowulf/tutorial/building.html*
- Jacek的 Beowulf 連結 *http://www.sci.usq.edu.au/staff/jacek/beowulf.*
- Beowulf安裝維護HOWTO *http://www.sci.usq.edu.au/staff/jacek/beowulf.*
- Linux平行計算HOWTO *http://yara.ecn.purdue.edu/~pplinux/PPHOWTO/pphowto.html*

5.3 相關論文

- Chance Reschke, Thomas Sterling, Daniel Ridge, Daniel Savarese, Donald Becker, and Phillip Merkey *A Design Study of Alternative Network Topologies for the Beowulf Parallel Workstation*. Proceedings Fifth IEEE International Symposium on High Performance Distributed Computing, 1996. *http://www.beowulf.org/papers/HPDC96/hpdc96.html*
- Daniel Ridge, Donald Becker, Phillip Merkey, Thomas Sterling Becker, and Phillip Merkey. *Harnessing the Power of Parallelism in a Pile-of-PCs*. Proceedings, IEEE Aerospace, 1997. *http://www.beowulf.org/papers/AA97/aa97.ps*
- Thomas Sterling, Donald J. Becker, Daniel Savarese, Michael R. Berry, and Chance Res. *Achieving a Balanced Low-Cost Architecture for Mass Storage Management through Multiple Fast Ethernet Channels on the Beowulf Parallel Workstation*. Proceedings, International Parallel Processing Symposium, 1996. *http://www.beowulf.org/papers/IPPS96/ipps96.html*
- Donald J. Becker, Thomas Sterling, Daniel Savarese, Bruce Fryxell, Kevin Olson. *Communication Overhead for Space Science Applications on the Beowulf Parallel Workstation*. Proceedings,High Performance and Distributed Computing, 1995. *http://www.beowulf.org/papers/HPDC95/hpdc95.html*
- Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, Charles V. Packer. *BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION*. Proceedings, International Conference on Parallel Processing, 95. *http://www.beowulf.org/papers/ICPP95/icpp95.html*
- Papers at the Beowulf site *http://www.beowulf.org/papers/papers.html*

5.4 軟體

- PVM - Parallel Virtual Machine *http://www.epm.ornl.gov/pvm/pvm_home.html*
- LAM/MPI (Local Area Multicomputer / Message Passing Interface *http://www.mpi.nd.edu/lam*
- BERT77 - FORTRAN conversion tool *http://www.plogic.com/bert.html*
- Beowulf software from Beowulf Project Page *http://beowulf.gsfc.nasa.gov/software/software.html*
- Jacek's Beowulf-utils *ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils*
- bWatch - cluster monitoring tool *http://www.sci.usq.edu.au/staff/jacek/bWatch*

5.5 Beowulf機器

- Avalon 是由 140台Alpha 處理器組成，36GB記憶體，可能是最快的Beowulf機器，計算速度高達47.7Gflops，在全世界前五百快的機器中排名第114。 *http://swift.lanl.gov/avalon/*
- Megalon-A Massively PArallel CompuTer Resource (MPACTR)由14台個人電腦組成，每台電腦內有四顆Pentium Pro200處理器，總共有14GB記憶體 *http://megalon.ca.sandia.gov/description.html*
- HIVE - Highly-parallel Integrated Virtual Environment 是另一套高速的Beowulf超級電腦，有64個計算節點，共計128顆處理器，4GB記憶體。 *http://newton.gsfc.nasa.gov/thehive/*
- Topcat 是一套比較小型的機器，總共有16顆處理器和1.2GB記憶體。 *http://www.sci.usq.edu.au/staff/jacek/topcat*
- MAGI cluster 是個有趣的網站，內有許多有趣的連結。 *http://noel.feld.cvut.cz/magi/*

5.6 其他有趣的網站

- SMP Linux *http://www.linux.org.uk/SMP/title.html*
- Paralogic - Buy a Beowulf *http://www.plogic.com*

5.7 歷史

- Legends - Beowulf *http://legends.dm.net/beowulf/index.html*
- The Adventures of Beowulf *http://www.lnstar.com/literature/beowulf/beowulf.html*

6 原始碼

6.1 sum.c

```
/* Jacek Radajewski jacek@usq.edu.au */
/* 21/08/1998 */

#include <stdio.h>
```

```
#include <math.h>

int main (void) {

  double result = 0.0;
  double number = 0.0;
  char string[80];


  while (scanf("%s", string) != EOF) {

    number = atof(string);
    result = result + number;
  }

  printf("%lf\n", result);

  return 0;

}
```

6.2 sigmasqrt.c

```
/* Jacek Radajewski jacek@usq.edu.au */
/* 21/08/1998 */

#include <stdio.h>
#include <math.h>

int main (int argc, char** argv) {

  long number1, number2, counter;
  double result;

  if (argc < 3) {
    printf ("usage : %s number1 number2\n",argv[0]);
    exit(1);
  } else {
    number1 = atol (argv[1]);
    number2 = atol (argv[2]);
    result = 0.0;
  }

  for (counter = number1; counter <= number2; counter++) {
    result = result + sqrt((double)counter);
  }
```

```
  printf("%lf\n", result);

  return 0;

}
```

6.3 prun.sh

```
#!/bin/bash
# Jacek Radajewski jacek@usq.edu.au
# 21/08/1998

export SIGMASQRT=/home/staff/jacek/beowulf/HOWTO/example1/sigmasqrt

# $OUTPUT must be a named pipe
# mkfifo output

export OUTPUT=/home/staff/jacek/beowulf/HOWTO/example1/output

rsh scilab01 $SIGMASQRT         1  50000000 > $OUTPUT < /dev/null&
rsh scilab02 $SIGMASQRT  50000001 100000000 > $OUTPUT < /dev/null&
rsh scilab03 $SIGMASQRT 100000001 150000000 > $OUTPUT < /dev/null&
rsh scilab04 $SIGMASQRT 150000001 200000000 > $OUTPUT < /dev/null&
rsh scilab05 $SIGMASQRT 200000001 250000000 > $OUTPUT < /dev/null&
rsh scilab06 $SIGMASQRT 250000001 300000000 > $OUTPUT < /dev/null&
rsh scilab07 $SIGMASQRT 300000001 350000000 > $OUTPUT < /dev/null&
rsh scilab08 $SIGMASQRT 350000001 400000000 > $OUTPUT < /dev/null&
rsh scilab09 $SIGMASQRT 400000001 450000000 > $OUTPUT < /dev/null&
rsh scilab10 $SIGMASQRT 450000001 500000000 > $OUTPUT < /dev/null&
rsh scilab11 $SIGMASQRT 500000001 550000000 > $OUTPUT < /dev/null&
rsh scilab12 $SIGMASQRT 550000001 600000000 > $OUTPUT < /dev/null&
rsh scilab13 $SIGMASQRT 600000001 650000000 > $OUTPUT < /dev/null&
rsh scilab14 $SIGMASQRT 650000001 700000000 > $OUTPUT < /dev/null&
rsh scilab15 $SIGMASQRT 700000001 750000000 > $OUTPUT < /dev/null&
rsh scilab16 $SIGMASQRT 750000001 800000000 > $OUTPUT < /dev/null&
rsh scilab17 $SIGMASQRT 800000001 850000000 > $OUTPUT < /dev/null&
rsh scilab18 $SIGMASQRT 850000001 900000000 > $OUTPUT < /dev/null&
rsh scilab19 $SIGMASQRT 900000001 950000000 > $OUTPUT < /dev/null&
rsh scilab20 $SIGMASQRT 950000001 1000000000 > $OUTPUT < /dev/null&
```