循序遞迴平衡負載應用在 O(n^2) 時間複雜度問題

Cyclic-Recursive Load Balancing on O(n^2) Computation

周朝宜　　張西亞　　陳德民　　鄭秀娟
國家高速網路與計算中心　　彰化縣彰德國中
{cychou, jerry, gavin}@ nchc.org.tw　　jeng@ctjh.chc.edu.tw

摘要

平衡負載（Load Balancing）一直是高效能計算（High Performance Computing）、分散式計算、叢集計算和工作排程(Job Scheduler) 最大的挑戰，本文提出循序遞迴（Cyclic-Recursive）、預測型循序遞迴（Predictinging Cyclic-Recursive）平衡負載理論應用在時間複雜度為 O(n^2)問題，兩者都優於循序平衡負載（Round Robin Algorithm），希望此項研究成果能對高效能計算、分散式計算、叢集計算工作者與系統管理者在工作排程上有所幫助。

關鍵詞：平衡負載, 高效能計算, 叢集計算

Abstract

Load balancing is a grand challenge problem on High Performance Computing, Distributed Computing, Cluster Computing, and Job Scheduler. In this paper, we implement new load balancing schemes, called Cyclic-Recursive and Predictinging Cyclic- Recursive on O(n^2) computational problem, both of them outperform related to an Round Robin algorithm. We hope this result will help users of High Performance Computing, Distributed Computing, Cluster Computing and system administrators improve job scheduling.

Keywords: Load Balancing, High Performance Computing, Cluster Computing

1. 前言

近年來，電腦計算環境已經走向叢集化、分散式計算及大型化，例如最近一次全世界超級電腦排名前五百名（top500）就有超過 70% 使用叢集系統[1]；又如國家網路與計算中心（以下簡稱本中心）正在籌建 15.974 TFLOPS（petaFLOPS, 每秒 1 千兆次的浮點運算）的大型叢集系統，總共有 2048 個計算核心(Cores) [2]。

在大型計算應用上如大氣象研究（Atmospheric Research）、計算流體力學（Computational Fluid Dynamics）、多尺度模擬（Multi-Scale Simulation）等，都需要大量的計算資源。

平衡負載的本質是讓所有的計算核心同時工作，平衡分配每個計算核心的工作量，以不要有閒置計算核心為原則，此關係著大型計算應用程式是否可以隨著計算節點增加而具有延伸性，也就是說，讓工作可以在最短時間內完成。

本文使用一個時間複雜度為 O(n^2) 的問題為例，提出循序遞迴（Cyclic-Recursive）、預測型循序遞迴（Predicting Cyclic-Recursive）平衡負載理論，並且與常用的循序平衡負載（Round Robin Algorithm）比較，實驗平台則使用本中心 Formosa 個人叢集系統[3]，希望此項研究成果能對高效能計算、分散式計算、叢集計算工作者與系統管理者在工作排程上有所幫助。

2. 研究方法與實驗平台

我們選用 Plouffe & Bellard 演算法 [4-6] 做為時間複雜度 O(n^2) 研究問題，以此方法求解圓周

率（π）之方程式如 Eq.(1)。

$$\pi = \sum_{k=1}^{\infty} \frac{k2^k}{\binom{2k}{k}} - 3 = \sum_{k=1}^{\infty} \frac{k\,2^k\,(\mathrm{k!})^2}{(2\mathrm{k})!} - 3 \qquad (1)$$

Bellard [5] 利用餘數定理(Chinese Remainder Theorem)與質數連乘技巧將 Eq.(1)轉換成算出任意 n 位數之連續 9 位值，其演算法如圖 1，從這個演算法可以發現計算時間是 O(n^2)，需要記憶體容量則固定。

```
N = Int((n + 20) log2(10))
sum = 0
For each prime number a with 2 < a < 2N {
   vmax = Int(ln(2N)/ln(a))
   m = a^vmax
   α = 0; v = 0; b = 1; A = 1
   DO k = 1, N
      b = k/a^v(a,k) b mod m
      A = (2k-1)/a^v(a,2k-1) A mod m
      v = v - v(a,k) + v(a,2k-1)
      IF (v > 0) {α = α + K b A^-1 a^(vmax - v) mod m}
   ENDDO
   α = α 10^(n-1) mod m
   sum = sum + α/m mod 1
}
fprintf ("%09d\n", (int)(sum*1e9))
```

圖 1. Plouffe & Bellard Algorithm for Calculating π

時間複雜度示意圖如圖 2。

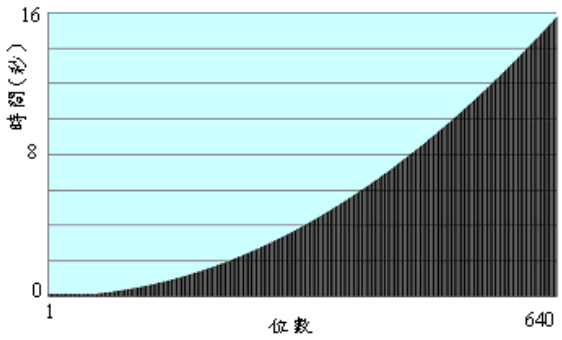

圖 2. 時間複雜度

在平行處理之工作切割方式以循序安排（Round Robin Algorithm）最常見，以下稱為“Round Robin”， 依照計算節點順序分割，例如要計算 n 位則第一個節點負責計算 1~(n/總節點數)、第二個節點負責計算(n/總節點數)+1~2(n/總節點數)…以此類推，例如使用 4 個節點計算 108 位之安排方式，計算節點編號 0, 1, 2, 3 分別負責計算 1~27、28~54、55~81、82~108 位數，如圖 3，從圖 2 時間複雜度圖很容易推論此方法會造成負載不均衡的情形。

計算節點編號	負責計算位數
0	1-27
1	28-54
2	55-81
3	82-108

圖 3. Round Robin Load Balancing

我們提出新的負載平衡分配方式來改善此情形，稱為循序遞迴（Cyclic-Recursive），依照循序遞迴方式安排工作，例如使用 4 個節點計算 108 位之安排方式，位數 1~9 由計算節點 0 負責計算、位數 10~18 由計算節點 1 負責計算、位數 19~27 由計算節點 2 負責計算、位數 28~36 由計算節點 3 負責計算、位數 37~45 也由計算節點 3 負責計算、位數 46~54 由計算節點 2 負責計算、位數 55~63 由計算節點 1 負責計算、位數 64~72 由計算節點 0 負責計算、位數 73~81 也由計算節點 0 負責計算、位數 82~90 由計算節點 1 負責計算、位數 91~99 由計算節點 2 負責計算、位數 100~108 由計算節點 3 負責計算，如圖 4。

計算節點編號	負責計算位數		
0	1-9	64-72	73-81
1	10-18	55-63	82-90
2	19-27	46-54	91-99
3	28-36	37-45	100-108

圖 4. Cyclic-Recursive Load Balancing

NCHC Formosa PC Cluster 是我們的實驗平台，它採用 IBM xServer x335 主機板，擁有 170 個 Intel Xeon 計算節點（2.8 與 3.0 GHz），總共有

340 個中央處理器，每個節點配有 2 至 4 *GB* DDR266 記憶體與兩個在主機板上（onboard）Gigabit Ethernet 網路卡面，透過 8 個 48 port 的 Nortel Baystack 5510 切換器(Switch)串接在一起。

作業系統為 GNU/Debian (2.4.26 SMP) Linux, 提供編譯器有 GNU GCC Collection 3.04、Intel C/C++ 8.0、PGI C/C++ 5.2 等，MPI-Middleware 有 MPICH、LAM/MPI、HP-MPI 等，使用 Scalable OpenPBS 1.0.1 佇列系統(Queuing System)、Maui 3.2.5 排程軟體（Scheduler）。我們採用 Intel 編譯器搭配 MPICH，做為模擬實驗的基礎。

3. 結果分析

表 1 是使用 4 個中央處理器計算 6048 位數，每個中央處理器所需時間，以秒為單位，從表中數據可以發現 "Cyclic-Recursive" 切割方式比 "Round Robin" 切割方式獲得較佳的負載平衡，Round Robin 方式讓計算節點編號 0 的中央處理器有近 1988 秒的閒置時間（Idle Time）。

表 1. The Elapsed Time in seconds of Round Robin *vs.* Cyclic-Recursive for calculating 6048 digits on 4 processors

Proc id	Round Robin	Cyclic-Recursive
0	65	907
1	425	907
2	1080	906
3	2053	906

表 2 是 "Cyclic-Recursive" 切割方式計算 6048 位數所需時間，以秒為單位、程式加快速度(Speedup)、程式效能(Efficiency)，從表中數據可以發現使用 4、16 個中央處理器可以獲得 4、16 倍之加快速度，程式效能都是 1。

使用 64 個中央處理器時可以獲得近 56 倍之加快速度，程式效能為 0.87，這是因為 6048 位數無法平均安排至每個中央處理器。

對於此類問題，我們提出預測型循序遞迴工作安排模式，以下簡稱 "Predicting Cyclic-Recursive"，透過預估模式將前面負載小的工作收集，而轉換成計算節點個數的整數倍。

表 2. The Elapsed Time in seconds, Speedup, and Efficiency of the Cyclic-Recursive Load Balancing on calculating 6048 digits

No. of Proc(s)	Elapsed Time (Sec.)	Speedup	Efficiency
1	3638	1	1
4	907	4.01	1
16	227	16.03	1
64	65	55.97	0.87

經過曲線湊配法（curve fitting），我們得到其時間複雜度函數為 Eq.(2)。

$$t = 3.50\times10^{-5}x^2 - 6.65\times10^{-5}x + 2.76\times10^{-5} \quad (2)$$
$$, x = 1,2,3,...$$

從圖 1 演算法可以知道此計算 π 小數點後位數，以九位數為一計算組合， 欲計算 6048 位數，必須分成 672 個計算組合，此無法平均分配至 64 個中央處理器處理，所以我們將前面 33 個工作依照 Eq.(2)合成單一個計算組合後，並且參照時間複雜度函數依序排列，轉換成 640 個計算組合，再依上述 Cyclic-Recursive 負載平衡方式安排工作。

表 3 是比較 Cyclic-Recursive 與 Predicting Cyclic-Recursive 負載平衡在使用 64 個中央處理器計算 π 小數點後 6048 位數，所需時間、程式加快速度、程式效能，結果顯示 Predicting Cyclic-Recursive 可以明顯改進 Cyclic-Recursive 負載平衡方式。。

表 3. The Elapsed Time in seconds, Speedup, and Efficiency of the Cyclic-Recursive and Predicting Cyclic-Recursive Load Balancing on calculating 6048 digits on 64 processors

	Elapsed Time (Sec.)	Speedup	Efficiency
Cyclic-Recursive	65	55.97	0.87
Predicting Cyclic-Recursive	56	64.96	1.02

4. 結論與討論

從我們實驗數據可以發現我們所提出的循序

遞迴(Cyclic-Recursive)和預測型循序遞迴(Predicting Cyclic-Recursive)方法都比循序(Round Robin)方法，獲得較佳的負載平衡、程式加快速度(Speedup)、程式效能(Efficiency)，在使用 4、16 個中央處理器處可以獲得 4、16 倍之加快速度，程式效能都是 1，表示 6048 位數計算可以平均安排至每個中央處理器，此方法可以獲得很好的延伸性。

Cyclic-Recursive 使用 64 個中央處理器時，6048 位數無法平均安排至每個中央處理器，導致程式的加快速度為 56，程式效能為 0.87，Predicting Cyclic-Recursive 透過時間複雜度函數將前面負載小的工作收集，而轉換成計算節點個數的整數倍，獲得 64 倍之加快速度，程式效能是 1，明顯改善負載平衡。

我們希望此項研究成果能對高效能計算、分散式計算、叢集計算工作者與系統管理者在工作排程上所所幫助。

參考文獻

[1] Top 500 Homepage, http://top500.org

[2] NCHC Homepage, http://www.nchc.org.tw

[3] NCHC Formosa PC Cluster Homepage, http://formosa.nchc.org.tw

[4] S. Plouffe, “On the computation of the n’th decimal digit of various transcendental numbers,” November 1996.

[5] F. Bellard, “Computation of the n’th digit π of in any base in O(n^2),” January 10, 1997

[6] Fabrice Bellard's PI : http://fabrice.bellard.free.fr/pi