簡介:

多相流作為計算流體力學的一個分支已有許多年,許多物理現象都屬於多相流之範疇,舉凡氣泡、雨、波浪、泡沫、噴泉等等皆是,其中液滴碰撞為一個非常基礎的物理現象。

液滴碰撞之應用可以說是相當的廣泛,常應用於雨滴成核機制等氣象學相關的領域,而另一方面在噴霧過程中,液滴碰撞扮演著重要的角色,尤其是發生在有體積限制的內燃機,液滴碰撞學對於內燃機內部油滴分佈的了解有很大的幫助,因為油滴散佈的情況會直接影響到引擎的燃燒效率,因此對於引擎效率的改善有相當的貢獻。

傳統工業上,許多設備也均以噴霧方式引入工作機制,如噴墨列印技術、噴霧冷卻技術及噴霧成形技術等。

電子產業上,運用金屬噴射技術可快速將焊材、合金、低熔點的金屬塗抹於晶圓、陶瓷及另外可焊接之基板上。

生醫產業上,液滴噴霧技術常運用在醫療器材的噴塗技術上,如許多醫療設備為了避免感染、血凝或組織創傷等,會利用噴霧技術將抗菌劑噴灑於醫療設備上。又如將微液滴應用於超微量 DNA 接合反應,以節省試劑的使用量。

因此我們可以得知,瞭解液滴形成的機制與其碰撞之後會產生的物理現象,有助於我們對於液滴碰撞會有更多的了解,因此以電腦程式模擬其產生之物理現象,就成為一種方便了解,且經濟實惠的一種手段。早期使用數值模擬複雜的物理問題時,為了運作龐大的計算量,往往需要高速的超級電腦或是工作站,然而超級電腦的成本昂貴所費不貲,資源無法遍及一般民眾使用,而計算量的需求卻更是有增無減。

隨著科技的發展,快速節能的電腦推陳出新,伴隨著價錢逐步下降,個人電腦已走進普羅大眾的生活之中,成為人人皆可消費的需求。

於是平行處理的概念應運而生,可將複雜的物理問題分割分別求解,再輔以網路傳輸將運算的結果串聯在一起,這類運算裝置名為叢集電腦(Cluster),因此由個人電腦串接成的叢集電腦就成為目前甚為普遍的平行計算工具。叢集電腦能以較低的價格提供龐大的計算能力,因此成為高效能計算機的主流架構。然而在電腦叢集系統上執行的平行程式不易撰寫,且資料輸入/輸出與網路通訊消耗的資源時間使整體系統效能嚴重受到影響。

適用於叢集電腦的平行演算法為區域切割法(domain decomposition method),此方法可使處理器的效能更有效的發揮,減少網路傳輸所耗費的時間,更可以依據物理問題與計算的處理器數量採取不同的分割法,來達到最好的效率。

本研究使用時間步進法(fractional time step method)求解非穩態的 Navier-Stokes 方程式,並將其應用在面追蹤法(Front-Tracking method)計算液滴碰撞之課題,可減少使用疊代法計算花費的時間,且此套演算法更可運用於模擬多數流體力學問題。

問題描述

面追蹤法（Front-Tracking）最明顯的特徵是把界面獨立於固定網格之外，界面是可移動的，而固定網格是為投射方法（projection method）來計算流場之用，在程式中，界面是由許多點所構成的曲線，主要由兩種物理量表達，一種是此點座標，另一種是在此點的物理性質，而將點和點的連接程曲線的方式有很多種，常用的兩種內插方式就是Lagrange interpolation和Legengre interpolation，而表面張力的影響可藉由計算介面曲線的curvature(如2.1式的末項)來得到，再把表面張力造成的影響以權重的方式散佈到鄰近界面的固定網格上做計算，而介面的移動也是藉由鄰近界面之網格上的速度來決定，造成固定網格與介面這兩個獨立東西有交互影響的作用，如圖1所示。

本論文使用的程式為2D軸對稱，使用相同性質的兩個液滴做正向碰撞，因此，使用對稱的計算區域（domain）來減少計算量，所使用的計算區域為真實計算區域的四分之一。邊界條件的設定為：在行進軸上使用軸對稱，在液滴前進方向所對的邊使用對稱，給定液滴的初始位置 X_{int} 並利用重力場加速，在液滴加速於我們需要的速度時，並將重力場於程式中關掉，使液滴自由的碰撞，如圖2所示。

擬採用方法

本研究採用的面追蹤法(Front-Tracking)方法係由 Unverdi 與 Tryggvason 所開發，一般來說，求解Navier-Stokes 方程式，我們考慮的流場通常只為單相，而在多相流中，我們在計算不同相的不同材料性質時，必須考慮界面所造成的現象，例如表面張力對流場的影響，以下將推導加入界面項的統御方程式。由於界面項能夠藉由 δ 函數來表達。因此流體的運動能被假設由下式 Navier-Stokes 方程式所統御，我們利用液滴的初始速度 V0 （$2V_0 = Vr$）、液體的密度 ρl、與動壓 $\rho_l V_0^2/2$ 將 Navier-Stokes 做無因次化，而其中時間項t是利用 R/V_0 做無因次化，無因次化後的Navier-Stokes 方程式如下：

$$\frac{\partial(\rho\vec{v})}{\partial t}+\nabla\cdot(\rho\vec{v}\vec{v})=-\nabla P+\vec{g}+\nabla\cdot\frac{1}{\mathrm{Re}_l}\left(\nabla\vec{v}+(\nabla\vec{v})^T\right)-\frac{8}{We}\int_{\Delta s}\kappa\vec{n}\delta(\vec{r}-\vec{r}_s)da \qquad (2.1)$$

（2.1）式中 $\vec{v}$ 為速度向量、ρ 為密度、P 為壓力、$\vec{g}$ 為重力場、κ 為二次平均曲率、$\vec{n}$ 為方向朝外垂直於液滴表面的單位法向量、$\vec{r}_s$ 為介面（interface）中的空間向量。而式(2.1)中 Re_l 定義為 $\rho_l V_0 R/\mu$，We 定義為 $\rho_l V_r^2 D/\sigma$。式(2.1)

中的表面張力是藉由δ函數在單位體積中於不互相混和的介面（interface）積分值，目的是要計算整個液滴的表面所造成的力。

在模擬過程中，不可避免的一定要談到程式設計及執行。平行主機與其他電腦主機不同之處在於，平行主機本身擁有多個處理器(CPU)，因此如何設計程式使其能發揮平行主機的特色，這就是所謂的平行演算法(Parallel Algorithm)。
在做平行計算時，因平行主機本身擁有多個處理器，故在設計程式時就必須能將所有的資料，經過適度的分割，再丟到各個處理器去執行運算。此時，不但必須讓個別處理器的工作負荷一致，不會造成有的處理器還在工作，有的處理器已經做完或沒事做而等在一旁，否則就會延誤整個工作完成的時間。再者，因處理器的運算速度遠快於不同處理器間交換資料的速度或處理器到記憶體存取資料的速度，因此在設計程式時也必須將此點考慮在內。區域切割法(Domain Decomposition)是目前平行演算法中較為人常用者，因它能較易達到上述目的。而區域切割法有許多種切割的方式，常見的有只對X或是Y方向做的長條狀式的切割，也可以同時對X與Y方向做長方形式的對切。甚至為了因應許多複雜的幾何形狀，或許網格要做局部加密，可以將局部加密的之處與未加密之處分別切割。對於切割方式好壞之評估目前已有許多不同之研究，就不在此贅述。但切割的好壞評估方式，重點在於能不能使所平行的處理器負載是平衡的，最好是可以使所有處理器同時開始與同時結束，才不致浪費處理器的效能。
本研究使用的網格皆為垂直網格，並且會在變化激烈處加密，而所使用之區域切割法，為了簡明起見，直接對X方向作平均分配。
在正式開始修改主要的程式之前，首先必須要檢查撰寫平行程式的正確性，與驗證它的效能，於是我參考了G..Tryggvason的A short MATLAB Navier-Stokes Solver，這是用Matlab所寫的一個簡單的程式，用以解非穩態的Navier-Stokes方程式，總共只寫了46行，主要是以SOR（Successive Over-Relaxation）解Poisson方程式配合固定的交錯網格（staggered grid）為基礎。而求解的問題為穩態下的穴蝕流（driven-cavity flow)，我將其原程式改寫為MPI、OMP、MPI+OMP三個版本，分別驗證其準確性與效能，模擬結果如圖5。

在改寫成MPI平行程式時所搭配的幾個指令如下：
MPI_ISEND：不等待而將指定的資料立刻傳送到指定的處理器。
MPI_IRECV：接收指定的資料而且不經過等待。
MPI_ALLREDUCE：將不同處理器的同一個矩陣做全部相加或是取極值。
MPI_SENDRECV：同時發送與接收某筆資料。
MPI_BCAST：將某個資料廣播並且更新至所有處理器。
MPI_BARRIER：等待所有處理器傳送與接收完成再進行下一步。

改寫成 OMP 程式的方法，則是在迴圈開始前加上

!$OMP PARALLEL

!$OMP DO

結束後加上

!$OMP END DO

MPI+OMP 則是將每個節點視為一個計算單位，之間使用 MPI 來通訊，而每個節點裡面使用 OMP 來計算。

本文試著平行面追蹤法的程式,首先我們要探討大部分計算流體力學程式運算耗時最久的部分，於是我在我們想增加速度的程式上添加了一個計時器，計算每個副程式所需花的時間，如表 6，其中省略了程式的前處理部分，只討論大迴圈中的副程式。

循序(sequential)版程式流程圖如圖 6,其中的副程式分別如下 Diagnst、SetDt、Fregrid、AdvFrt、SetDens、Gadvect、Fcurv、AddSurfT、Gpressur、SetVisc，在這些副程式中，可以看出佔最多運算時間的副程式為 gpressur，也就是計算壓力的副程式，因此首先應該從這個副程式下手，而這個副程式最主要的部分也就是我們稱 SOR（Successive Over-Relaxation）的四方疊代的部分，但是傳統的 SOR 有所謂更新（update）方向的問題，如圖 7，造成無法切割區域來使用平行運算，若我們用傳統的 SOR，則會花上大量的時間由處理器 1 算完，傳到 2。如此這般完全沒有平行處理的效果，有可能比循序版的程式表現更差。

早在 1979 年之前，黑白點間隔法（red-black SOR method）或稱棋盤間隔法（checkerboard SOR method）就被提出了，而後 Loyce Adams 及 James M. Ortega 在 1982 年發表了使用顏色法（color scheme）的 SOR 用以平行計算，顏色法最多可以將網格分為四種顏色，其目的是為了使處理器可以向量化的計算，達到平行的效果。

為了達成可以平行運算的目的，吾人簡單的使用兩種顏色，將一般的 SOR 改成黑白點間隔法。具體而言黑白點間隔法是先計算所有的白色格點，然後計算所有的黑色格點，如圖 8 所示。也就是當 i+j 的值是偶數時為白色格點，i+j 的值是奇數時為黑色格點，計算黑色格點時其四周都是白色格點，而計算白色格點其四周都是黑色格點。因此，每一個黑色格點都可以單獨計算，白色格點也是如此。這一種計算方法是從原來的 SOR 演算法修改過來的，可能造成收斂速度的改變，以及最後各個格點數值的改變，但是卻較容易平行化。

由於黑白格點之間的計算彼此獨立，因此可以直接使用 OMP 將其迴圈平行化，而 MPI 的部份，必須要自己設定切割區域的邊界傳遞，可以以一個副程式將黑色格點算完之後存入一個一維矩陣，再將矩陣資料傳輸至左右兩旁，在傳輸時若使用無阻塞式(non-blocking)的傳輸法，將可以節省等待的時間，同理白色格點亦然。

計算完一次迴圈之後，需要在每個節點上檢查其誤差，之後使用集體傳輸指令將

誤差做加總，再分送到各個節點，以了解是否要繼續做下一次迴圈。若達需求之收斂狀態，必須將計算完畢之資料，再次蒐集、傳回主處理器，待主程式處理完無法平行的循序部份，在下次 SOR 過程之前，再將需要計算的資料廣播至所有處理器。

而本文模擬的案例為十四烷於一大氣壓空氣之液滴碰撞，由於彈開（bouncing）所需之計算量較少，因此本次模擬結合（coalescence）的狀況，模擬的結果大致如圖 9 所示。而又受限於國網中心平台有設定每個程式運算時間不可超過兩個禮拜，因此在較密網格的部份只能算到接近 1000 個 STEP，詳細計算的 STEP 如表 7 所示。

預期成效

我們將循序版的程式與平行版的程式，分別執行於六種不同大小的網格之下：128×128、256×256、512×512、1024×1024、1500×1500、2048×2048，並在其程式之間加入計時器，以測量其花費之時間，可得到如表 8 的五十四種結果，將成整理成圖 15 到圖 20，分別討論如下：

128×128：

如圖 15 所示，在此種網格之下，可以發現純 OMP 可以得到近乎線性，非常好的平行效率，而 MPI 與 MPI+OMP 則隨著處理器數目的提升得到的效果卻是越來越遞減的，可以得知由於格點數過少，分配給越多處理器來處理，所得到的計算時間就會不停的縮減，相對而言對於 MPI 的運算方式網路傳輸的時間所佔據的比例卻會不停的上升，因此時間就會耗費在網路傳輸上，而 SMP 架構的 OMP 則是未經過網路傳輸，直接由多核心的 CPU 對同一個節點的記憶體存讀取資料，因此可以發揮最佳的平行效率。

256×256：

如圖 16 所示，在此種網格之下，OMP 的優勢依舊存在。甚至超過線性的加速程度，這是不符合理論的，但若是對照圖 14，可以發現循序版的處理器的效能已經落在區域三（6×10^5）了，但是 OMP 兩個處理器或是四個處理器所分到的矩陣大小卻還能在區域二之中，因此不但單顆處理器速度加快，資料量又分開處理，因此其速度提升就會不尋常的高於其理論值。在 MPI 與 MPI+OMP 改寫方式的方面，可以發現在兩顆與四顆的時候，其速度提升都是線性的，但在八顆與十六顆的時候，由於單顆計算量再度下降，而網路傳輸資料更加繁忙，因此其速度提升會侷限在四倍左右就無法再次提升了。

512×512：

如圖 17 所示，在此種網格之下，MPI 與 MPI+OMP 的效能表現得到十分顯著的成長，因為循序版的疊代量隨著格點數的提升又變為之前的四倍，此時隨著分擔的處理器數量的增加，平行版的單顆處理器運算量相對於循序版而言也就大量的下

降了，而在網路傳輸方面，其佔據的時間相對於運算量的再度提升，就顯得與之前比來說，其比例變小許多，因此速度提升變的十分明顯。
但純 OMP 速度卻不增反減，甚至速度較循序版相比，變得差別不大，其原因我們可由 4.1 中得知，由於 512 的平方矩陣大小為 26 萬，遠遠超過圖 14 中範圍二最大的 10 萬，因此純 OMP 的效率十分的差。在看到 MPI+OMP 的部份，由於 OMP 運算量由 MPI 分配到二或四個節點上，因此在單個節點上 OMP 所負責的矩陣大小為 13 萬（兩個節點）或是 7.5 萬（四個節點），接近圖 14 中的範圍二，因此 MPI+OMP 的效率依然不輸純 MPI，近乎是線性的成長。

1024×1024：

當格點數達到 1024×1024 的時候，如圖 18，明顯可以看出以純 OMP 改寫的程式，平行效率遇到了瓶頸，其速度提升效率大約落在 1.6 到 1.8 附近，這與我們在 4.1 中 STREAM 所測得區域三中矩陣大小 10^8 的比值大約相符合，而 MPI+OMP 也因為就算將運算區域切成四個節點，每個節點 OMP 要負責的矩陣大小也超過圖 14 中區域二的上限 10^6，因此 MPI+OMP 的運算速度也達到了瓶頸，純 MPI 則因為 MMP 的獨立架構，每個運算單元都有各自的獨立記憶體與處理器可以使用，因此將 $1024^2 \div 16$，可以得知一個處理器分配到的矩陣大小是 65536，還可以落在圖 14 區域二中單個處理器效能極速下降之前，因此可以推估，在記憶體與處理器效率之極限尚未到達其瓶頸，但其速度仍然沒有上升成線性的效果，探討其原因，可以將其猜想或許因為網路傳輸達到了瓶頸，我們可以參考圖 21，測試出的網路環境告訴我們使用 Gigabit Ethernet 的網路環境，以 16 個處理器平行處理，其速度提升的極限大概約在 8~10 之間，我們無法得知當初他測試的是什麼樣的實例，而大多計算流體力學的應用程式顆粒是屬於細顆粒，也就是計算量/通訊量的比值較小，因此速度提升會受限於網路速度之延遲。

2048×2048：

如圖 23，可以看出由於矩陣過大的因素，MPI+OMP 在處理器到 16 個的時候速度提昇又再度下降了，我們可以得知 SMP 架構在矩陣過大的時候效能會急速下降，猜想其原因應該跟它們讀取同一塊記憶體有關係，因為同時存取同一塊記憶體，所以會造成讀取忙綠，而造成延遲的情況產生，而純MPI 由於獨立記憶體的架構，因此並不會同時讀取同一塊記憶體，而是分別存取各自的記憶體。

4.3 Front-Tracking 循序版與平行版程式之結果與討論

4.3.1 循序版與平行版準確性驗證

將改寫好的平行版程式與循序版程式，以比較軟體 Compare It! 分析在 nsteps 到三十萬步的時候密度輸出矩陣之誤差，如圖 22 所示，上下左右之數值完全相同，只有在中間變化較激烈的點，有最多小數點下兩位的誤差，由此可知運算過程中間幾乎沒有誤差，估計誤差產生在 MPI 的指令 MPI_ALLREDUCE 裡面，這個指令是傳送所有資料做集體通訊的運算，可能會有捨位誤差。

4.3.2 循序版與平行版計算結果之比較

吾人將 Front-Tracking 程式改寫為平行版本後，在每個副程式的開始前與結束後加上一個簡單的計時器，用以計算每個副程式分別耗費多少時間，由於知道程式中許多副程式是屬於不可平行的，而可以平行的地方，就稱為平行區域（Parallel region），而對於非固定計算量的程式而言，平行區域的比例是會因為計算條件的改變而改變，經過測試之後發現，網格在 128x128 的平行區域計算時間大約佔整體計算時間的 61%，在 256x256 之後則是都落於 81~99%，因此為了避免還要使用安達定律來轉換真的平行區域內的速度提升，本小節之討論結果只計算平行區域（Parallel region）的速度提升，而不做整體之速度提升。將各種不同網格數的結果成圖表討論如下。

128x128：

如圖 23，我們可以看出由於網格太少，除了 OMP 可以讓速度提升到兩倍以外，超過四個處理器之後，無論是 MPI 或是 MPI+OMP，所得到的速度提升效果都非常不好，甚至 MPI 使用八個以及十六個處理器的時候速度還會越來越慢，可以判斷出對於過少的網格，平行處理只有 OMP 可以達到理想的速度提升。

256x256：

網格數到了 6×10^5 左右，可以對照圖 14，我們可以得知矩陣大小在區域二內的 10^5~10^6 中間左右的位置，所以四個處理器用 OMP 方式之下的處理速度提昇大約是單個處理器的兩倍再多一點，而兩個處理器用 OMP 的速度並沒有快到兩倍，而是大約一倍多一點，所以在圖 24 中純 OMP 的部分與圖 14 得到的結果大致是相符合的，而在使用 MPI 與 MPI+OMP 方式來改寫的部份，我們先將先前簡單測試中的圖 16 與現在 256x256 中的結果圖 24 做比較，可以發現以 MPI+OMP 改寫的速度提升大致還是相同的，可是純MPI改寫的程式的速度提升卻是遞減的，探討其原因，可以發現或許是由於面追蹤法的程式在解壓力的時候多了需要設定邊界處的壓力，以及每個處理器必須檢查其計算區域與上一步的誤差，之後必須集體傳輸該區域誤差並且計算總值有無超過收斂標準，因此當處理器越來越多的時候，運算時間就會浪費在過多的通訊時間，造成以 MPI 改寫的程式的速度提升越來越小，反而以 MPI+OMP 改寫的由於將每個節點視做一個運算單位，而最多也只有四個節點，所以消耗在網路傳輸的時間較少，因此可以維持較佳的運算品質。

512x512：

結果如圖 25，可以看出已經超出純 OMP 改寫方式可以負擔的範圍，因此我們將重點放在以 MPI 與 MPI+OMP 改寫方式的變化，當 MPI 使用兩個處理器的時候，速度提升是接近三倍左右，但我們知道理論上最多只有兩倍，其原因可由之前 4.1 小節討論的結果，也就是因為單個處理器的速度已經到了圖 14 中的區域三，而用 MPI 改寫方式將工作分給兩個處理器，每個處理器分到的資料量還在圖 14 的區域二之中，因此處理的速度可藉由快取記憶體得到提升，而以純 MPI 改寫，分給四個處理器的狀況也是相同，其速度提升可以超過四倍來到五倍。但是到八個處理器或是十六個處理器之後， 純 MPI 程式的執行速度就開始下滑，可以推得

其原因是因為此時的資料傳輸量大於計算量,而是延宕在網路傳輸的時間太多導致平行效益並不高，反而更低。

1024×1024：

結果如圖 26,可以觀察出單單以 MPI 改寫的優勢已經超過同時使用 MPI+OMP 了，在這個格點數之下 MPI 的效能幾乎可以達到圖 27 中周朝宜等人所測得在國網中心的平台之下的最佳效能,可以得知此時的資料通訊時間已經相對於計算時間只佔了小部份，因此可以得到最高的速度提升與平行效益，而純 OMP 改寫方式由於先天的計算型態問題，因此當矩陣過大的時候，並沒有辦法得到好的平行效益，但是藉由合併 MPI+OMP 同時使用的形式,還是能因為 MPI 本身的優勢造成速度提升還是能線性的成長。

1500×1500：

結果如圖 28，可以發現奇怪的是 1500×1500 並沒有達到之前 1024×1024 中（圖 22）的速度提昇，整體而言曲線下降了一個層次，而這與我們推估的接下來隨著網格增加應該都可以達到最佳化的速度提升的推斷有所不符,對於這個奇怪的現象，可以用圖 14 來解釋，我們將 1500×1500÷8 可以得到每個處理器負責運算的矩陣大小為 2.8×10^{6}，這個數字會落在圖 14 的範圍 3 當中，因為格點數是 1024×1024 的時候，每個處理器分配到的矩陣大小，會小於 10^{6}，因此還能落在圖 14 的區域二之中,但是此時循序版的程式處理器的負載量就已經落在區域 3 之中了，所以若使用平行處理，恰好可以讓每一個處理器還能處於最佳化的負載量，而在 1500×1500 這個格點數的時候，就算使用了平行計算的方式，由於每個處理器分配到的資料量過大，所處的區域都落到圖 14 的區域三之中了，運算速度都已經進入較差的階段，因此此時的速度提升與平行效益就會看起來很差。

2048×2048：

執行結果如圖 29，速度提昇又回到我們所平台能達到的最佳效能，原因乃是因為此時的循序版程式所計算的資料量已經過大,循序版程式處理器的記憶體使用量已經嚴重超過最佳用量的上限了,因此處理器的運算時間都會耗費在記憶體與快取記憶體的存取之中，也就是說它的運算速度已經掉落到最差的階段，對照圖 14 來解釋，就是循序版與平行版程式都在圖 14 的區域三之中，但是循序版已經到圖三更後面的區域，計算速度會再度下降一個層次。反觀如果使用平行運算，可以有效的分散計算量，讓每一個處理器的負擔較小，自然可以得到較佳的效果。

我們將所有結果的平行效益，與其對應使用處理器的數量，將處理器的數量除以總網格大小，可以得到每個處理器分配到需要處理的網格大小，將其與對照的平行效益做成圖 30 的散佈圖的形式，我們可以發現在每個處理器所分配到的網格數在 10^{5} 以下的時候，以純 OMP 方式改寫的程式，得到的平行效益較佳，當超過 10^{5} 之候，以純 MPI 方式改寫的程式所得到的平行效益就會開始追上純 OMP 方式的，但之後純 OMP 方式的程式，平行效益就會開始遞減，而 MPI 與 MPI+OMP 方式改寫的程式的平行效益則會遞增,但整體而言以純 MPI 方式改寫的平行效益還是

會超過同時混合 MPI+OMP 的平行效益。
上述為在 Gigabit Ethernet 得到之結果，我們希望知道在高速網路 Infiniband 之下的效率為何。