國立臺灣大學理學院天文物理研究所

碩士論文

Institute of Astrophysics
College of Science
National Taiwan University

Master Thesis

使用高速圖形計算核心的
自適應網格精緻化的流體模擬
Adaptive Mesh Refinement Fluid Simulation with Ultra-fast Graphic Processing Units

蔡御之

Yu-Chih Tsai

指導教授：闕志鴻 博士

Advisor: Tzihong Chiueh, Ph.D.

中華民國 97 年 7 月

July, 2007

致謝

能夠完成這篇論文，最感謝的就是我的指導老師 闕志鴻教授，平時不厭其煩的解決學生們的問題，有一次甚至為了我的問題，在學校待到凌晨三點。這幾天為了準備論文口試，平常沒能理解的問題洶湧而來，同樣的問題請教老師多次，老師仍然細心的替學生講解。試問，在學校裡能有多少老師會願意這樣為學生犧牲奉獻呢？

能夠完成這支程式，第一個要感謝的就是薛熙于學長。小魚學長幫我完成了最困難的 GPU Solver，這其實是程式裡最重要的地方，平常也常常替我解答許多程式方面的問題。如果沒有他，我想我的畢業時日肯定遙遙無期。再來要感謝的是超級照顧學弟們的胡德邦學長，超有義氣的阿邦邦，在自己忙的昏天暗地的時候，仍然願意花許多時間解答眾多學弟們的問題。如果沒有他，我學習 FFT 所花費的時間恐怕就要多上兩倍，在最重要的六月，學長犧牲了自己的時間指導我，讓我順利的了解使用 FFTW 的方式。再來是神人般的黃承光學長，感覺世界上似乎沒有任何問題難得倒小光學長。每次想了半天都無解的問題，小光學長總是有辦法隨口說出讓人意想不到的答案。還有陳志清學長，Sid 帶來的歡樂氣氛與美食，真是讓人捨不得畢業。還有為實驗室機房無怨無悔付出的簡嘉宏學長，沒有簡嘉宏學長，機房裡的電腦不會穩定的工作，更不會有這麼多的模擬成果。還有親切的滕曉峯學長，在我心情低落時，給予我不少的指導與安慰，Steven 學長認真負責的工作態度，絕對是大家學習的好榜樣。還有總是一早就到 801 努力工作的簡鴻裕學長，在實驗室裡日夜顛倒的工作習慣中，簡鴻裕學長是唯一的一股清流。還有已經畢業的林凱揚學長與林俐暉學姊，這對賢伉儷在實驗室裡的佳話一定會一直流傳下去。最後是在學習路上互相扶持的實驗室夥伴們，陳彥麟、吳尚憲、詹弘旭、李孟修，還有已經畢業的吳靜澄與湯郁然，有了你們實驗室歡樂氣氛不斷，我會懷念著和你們在一起的日子。特別提出來的是姚慧敏小姐，您才是實驗室裡最重要的角色，沒有你，大家都拿不到錢了 XD。妳給予我的幫助和鼓勵，絕對是雪中送炭最好的例子。

最後一定要感謝我親愛的父母、兩位哥哥、與在我身旁陪伴我的小獅子，有你們的支持，我才可以順利的完成這篇論文。

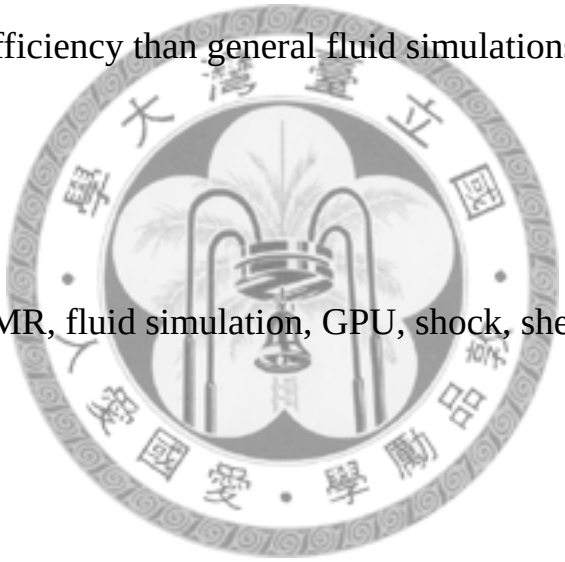
摘要

自適應網格精緻化是一個當計算誤差變大時，將網格切得更細密以達到更高精確度的一個模擬方法。本論文發展一個使用高速圖形計算核心的自適應網格精緻化的模擬，提供可達一般自適應網格流體模擬的十倍效能。

關鍵字：自適應網格、流體模擬、爆炸波、切變流、震波、顯示卡。

Abstract

Adaptive Mesh Refinement is an idea of saving computing resources. In many cases the most important flow features occupy only a small region in the computation domain. AMR will recursively refine grids when errors grow and remove the fine grids where the coarser grids still give an enough accuracy. In my thesis, I present a fluid simulation program using the graphic processing units which are much powerful than the central processing unit in common personal computers. The program offers ten times computing efficiency than general fluid simulations before.



Keywords: AMR, fluid simulation, GPU, shock, shear flow.

目　錄

圖 目 錄

表 目 錄

1. 緒 論

在探討天文大小尺度的結構形成與天文動力系統分析的問題，一定不能避免的會遇到非線性流體的演化。非線性問題大多無法由理論計算完整的解出，目前最普遍的解決方式，都是使用電腦數值模擬。為了精確的模擬系統的演化，常常會需要極高的解析度，也就是說流體模擬極為需要計算能力與記憶體大小。但是在許多流體模擬問題裡，真正重要的精細結構通常只佔模擬空間的小部分。如果為了要將精細結構模擬出來，而讓整個空間的網格點都切得很細，是件非常浪費計算資源的事情。

自適應網格調變(Adaptive Mesh Refinement, AMR)是個可依使用者需求將網格的精細度做即時地調變(refine)的演算法，是由 M.J.Berger 和 J.Oliger 在 1984 年[1]所發展出來的。AMR 對於任何使用者感興趣的區域切得十分細密，讓這些區域能夠以更高的解析度進行精確地運算。而其他平凡無奇的區域則用較粗的網格計算，節省計算資源與提高程式運算的效能。

本程式使用 AMR 技術模擬流體的演化，配合上超高速的顯示卡運算核心(Graphic Processing Units, GPUs)，提供了強大的運算效能，約是使用一般電腦中央處理器(Central Processing Unit, CPU)運算之 AMR 流體模擬程式的十倍運算效能。這主要是由於現今顯示卡的計算能力，已經遠遠的超過了電腦的中央處理器。以現今最快的顯示卡與中央處理器相比，目前(2008/07)市面上銷售的個人電腦中央處理器最快的是 Intel QX9650，其理論計算能力是 96 GigaFLOPs (FLoating point OPerations per second, 每秒可執行的浮點運算個數)。而最新的顯示卡 NVIDIA GeForce GTX 280 則擁有 933 GFLOPs 的運算能力，約是 QX9650 的 10 倍。

以本實驗室所使用的 NVIDIA GeForce 8800 GTX 顯示卡為例，這張卡是 2007 年最強大的顯示卡，擁有 128 個串流處理器(Stream Processors)，每個串流處理器的運作時脈是 1.35 Ghz。而每個串流處理器可在一個時脈週期裡同時運算一個浮點數的乘法與一個浮點數的加法，於是這張顯示卡的浮點運算能力是$128 \times 1.35 \times$

2 = 345.6 GFLOPs，仍然擁有現今最快的中央處理器的 3.5 倍運算效能。

以價格來說，一台插上兩張 NVIDIA GeForce GTX 280 顯示卡的個人電腦售價約是新台幣五萬元，其擁有的浮點運算能力是 2 Tera-FLOPs，這樣的性價比是以往任何超級電腦的數百倍以上。顯示卡擁有強大的運算能力與平民化的價格，絕對是未來建造平行運算電腦叢集的首選。

在本論文的第二章，我們將介紹這個流體模擬程式的主要架構與重要的細部資訊。第三章我們將實際模擬兩個常見的流體模擬程式的測試，與流體方程式所計算出的理論解相比，了解本程式的可信度。並且我們同時測試本程式的效能表現，了解使用顯示卡運算帶來多少倍的效能增進。第四章我們實地進行切變流(Shear Flow)的模擬，分析切變流所產生的紊流(Turbulence)系統，並與近期其他研究團隊的分析結果相比，了解本程式的流體演化核心(Solver)在紊流系統的可靠度。

2. 程式內容介紹

在本章我們將對程式的內容做重點介紹，2.1 節我們介紹程式使用的語言與資料結構，2.2 節我們介紹程式執行的流程，2.3~2.7 節我們依序講解程式重點函式的主要內容。

2.1 程式簡介

本程式使用 C++語言，呼叫顯示卡運算的程式碼使用 NVIDA 提供的 CUDA (Compute Unified Device Architecture)[2]介面。CUDA 是一個以 C++語言為基礎的操作介面，是 NVIDIA 近期為了通用顯示卡運算(General Purpose GPU)市場編寫出來的。在早期，通用顯示卡運算的發展十分困難，當時顯示卡雖然擁有強大的計算能力，但由於操作介面複雜與通用性不佳，並沒有受到太多人的重視。當時使用顯示卡作科學計算的方式，都是想盡辦法讓所有的運算轉型成為繪圖的格式，讓顯示卡誤認為這些運算都是為了畫圖而做，這使得程式的編寫十分困難且沒有效率。直到 CUDA 推出，才讓許多使用者注意到通用顯示卡運算發展的可能性。CUDA 讓使用顯示卡運算變得像是學習一個類似 C++語言一樣方便，對於原本就精通程式語言的人來說，實在不算是一個門檻。本程式的流體演化核心就是用 CUDA 編寫而成，讓顯示卡取代中央處理器作許多複雜且耗時的運算，大幅地加速流體模擬的效率。

本程式的流體演化演算法參考 Hy Trac, Ue-Li Pen, 2002, A Primer on Eulerian Computational Fluid Dynamics for Astrophysics [3]，其使用尤拉方程式演化流體並確保總變化衰減(Total Variation Diminishing, TVD)。

尤拉方程式：

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \vec{\nabla} \cdot (\rho \vec{v}) = 0, & (1) \\ \dfrac{\partial (\rho \vec{v})}{\partial t} + \vec{\nabla} \cdot (\rho \vec{v} \vec{v}) + \vec{\nabla} P = 0, & (2) \\ \dfrac{\partial e}{\partial t} + \vec{\nabla} \cdot [\,(e + P)\,\vec{v}\,] = 0, & (3) \end{cases}$$

其中 ρ 是流體密度、t 是時間、$\vec{v}$是流體的流速、P 是流體壓力、e 是流體的能量密度，包含了熱動能與流體動能。

所以空間中每個格點都存有五個變量，分別是 ρ 、ρv_x 、ρv_y 、ρv_z 、與 e。而壓力可由理想流體的狀態方程式算出，

$$P = (\gamma - 1) \cdot \left(e - \frac{1}{2} \rho v^2 \right), \qquad (4)$$

這裡的γ是流體定壓比熱與定容比熱的比值。

空間總變化(Total Variation)的定義是

$$TV(u^t) = \sum_{i=1}^{N} |u_{i+1}^t - u_i^t|, \qquad (5)$$

代表著空間中每個格點與相鄰的格點間的差值的絕對值的總合。可以改寫成另一個形式

$$TV(u^t) = \sum \mathrm{u}_{\max} - \sum \mathrm{u}_{\min}, \qquad (6)$$

這代表著空間中所有相對極大值之和與相對極小值之和的差。這樣確保所有的數值誤差不會造成任何非物理的現象，可視為自然的能量消散，由動能轉變為熱能。

本程式是個三維的流體模擬程式，空間切割方式參考 Paramesh[4]，將空間切成多個不同大小的長方體，我們稱這些長方體作為小片(Patchs)，每個小片裡都擁有Nx × Ny × Nz個格點(Cells)。不同大小的小片差別在於格點更細密，而不是小片內擁有的格點數不同。我們把不同大小的小片分成不同階級(Level)，階級越大代表著網格越細密。

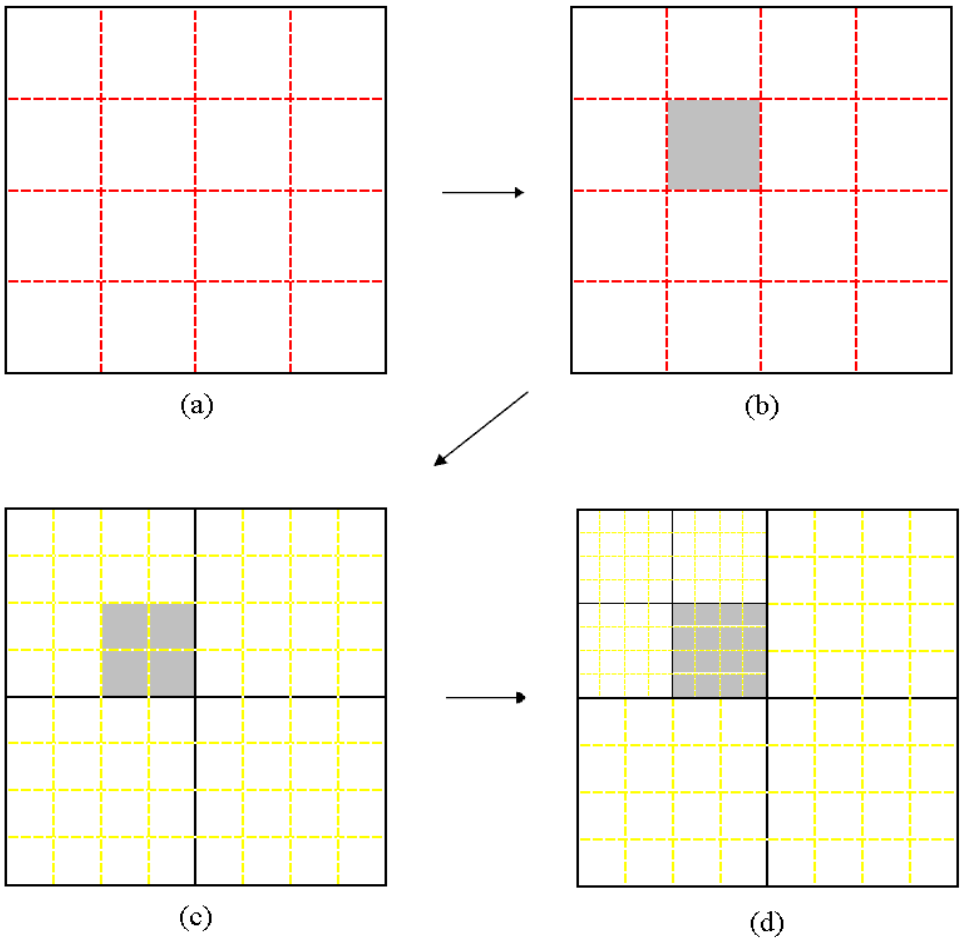


圖 2.1： 空間網格分佈示意圖

參考圖 2.1a，這是一個二維的例子，圖中是一個4 × 4的小片，每一個紅框是一個網格。首先我們假設整個模擬空間只有這個階級為零(Level = 0)的小片，如果我們發現圖上灰色格點解析度不足，如圖 2.1b，希望用更高解析度的網格處理。在本程式的架構下，我們就會產生四個階級一的小片，如圖 2.1c。我們的方法是，只要整個小片中任何一點被認定為解析度不足，那麼整個小片都會被切細，而不是只有被判定的那個點，因為本程式以一個小片作為基本的調變結構。同時我們提高解析度的步驟，是將每個網格點在各個維度都砍半，在二維裡就會產生四個大小相同的細網格。而不論粗或細的小片裡，都擁有一樣數目的網格點。如果我們依然覺得灰色區域解析度不夠，把網格繼續切細，就會產生四個階級二的小片，如圖 2.1c。此時空間中總共有一個階級零的小片、四個階級一的小片、以及四個

階級二的小片，階級零與階級一的小片並不會因為被切細而被抹除。我們會把這個階級零的小片稱作階級一小片的父小片，反之階級一的小片就是階級零小片的子小片。而這些相同階層且相鄰在一起的小片們，則稱作為彼此的鄰小片。鄰小片間不一定要擁有同樣的父小片，只要大小相同且相鄰就是彼此的鄰小片。

有時候我們會希望限制程式最粗的解析度，而不是由階級零的小片開始。在程式裡可以設定最低階級，如果設為三，意味著空間中最粗的小片就是階層三，且空間中擁有 8^3 個階級三的小片。如果一個小片大小是 4^3，那麼代表空間中最粗的網格總數是$8^3 \times 4^3 = 32^3$。

由於模擬機器的記憶體大小有限，不能讓網格無止境的切割下去。在程式裡可以限制總網格的數目，以及最細網格的階級。超過限制就不允許程式繼續切細產生更多的小片，除非有其他不需要的小片被移除。

2.2 程式流程

圖 2.2 是本程式的主程式虛擬碼，十分精簡易懂。在程式執行的一開始，使用 Initialize 函式裡把所有初始條件讀入，並經過初始化網格調變(Refinement)，成為程式可以直接進行演化的格式。或是使用 Load_Data 函式讀入之前模擬的備份檔，並轉化為本程式可以直接演化的格式。接下來的 for 迴圈裡面，分別是計算適當時距的大小(Determine_dt)，以及進行演化(Intergration)的函式。這個虛擬碼裡最複雜的函式就是 Integration，我們特別拿出來做更進一步的講解。

```
int main()
{
    Initialize()  or  Load_Data();

    for(int counter =0 ; counter < limit ; counter++)
    {
        Determine_dt();
        Intergration();
    }

    return 0;
}
```

圖 2.2： 主程式虛擬碼

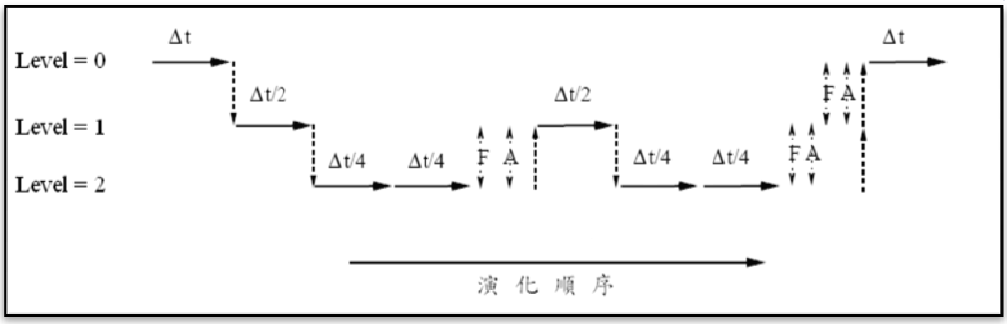


圖 2.3： 演化流程圖

圖 2.3 是 Integration 的流程圖，在這個例子裡只有三個階級的網格點。首先先從粗網格開始演化，然後依序是細網格、最細網格，演化到沒有更細的網格為止。而其演化時距會因為網格粗細的不同而改變，因為粗細網格的資訊傳遞速度約略

相同，對於不同的網格寬度必須選用不同的演化時距，否則會因資訊傳遞距離超過網格寬度造成錯誤的演化。由於空間的長度是以二等分做切割，於是我們也選用二等分切割演化時距。演化到了最細的網格點，繼續將最細的網格再次推進，直到跟上一層的網格點同步。粗細網格同步後，我們用細網格的資訊修正粗的網格，因為我們相信細網格演化具有較高的正確性，這就是圖上的 F 步驟。修正之後，我們用當下的資訊調變細網格的分佈，將需要精細網格的區域保留或新建、不需要的移除，這就是圖上的 A 步驟。此時第一、二階級的網格已經同步且重新整理過，才繼續演化階級一的網格，使其與階級零的網格同步。不過此時並不急著向上修正，因為最細網格的資訊還未被演化出，直到最細網格也同步後，再一次向上修正到階級零，並分別調變階級一與階級二的網格分佈。如此我們就完成了一個 Integration 的函式，將流體演化了Δt時間。

當我們在進行細格點演化時，在粗細邊界上我們會對粗格點插分，得到細格點所需的邊界條件。細格點第二次演化時，這些邊界資料必須由兩個不同時刻的粗格點作時間的插分得到，所以我們選擇對粗格點先演化。也因此實際上每個小片裡存有兩組變數的值，分別是演化前後的資訊，以便需要時能夠作時間插分。

2.3 模擬初始條件的賦值

程式的初始化動作由 Initialize 函式完成，內容主要分為兩大部分，第一個是對空間上每個格點賦予初始值，第二個則是將初始值做網格粗細調變。調變後產生更細的格點，再賦予新的格點初始值，兩個步驟反覆運行，直到每個階層的網格調變都完成且都賦予初始值，才進入演化的階段。

賦予初始值有幾種不同的方式，最簡單的一種是直接定義一個空間函數。不管粗或細的網格點，都有其相對應於空間的座標，由格點座標決定變量的值。此時細格點的變數值，是由空間函數決定，而非插分而來。然而有時候我們定義的初始值，是由先前其他非 AMR 的模擬程式產生，這時候我們有明確定義變量的網格點，可能只有 256^3、512^3、或者其他固定的解析度。此時我們就需要用插分來產生比原來還細的格點上的變數值，或是將格點的值取平均給予較粗的格點。

2.4 網格調變

調變網格粗細(Refinement)是 AMR 程式架構的核心，在每次粗細網格演化同步時，我們都可以進行網格粗細的調變，隨時都讓程式擁有最適當的網格分布。不過調變網格需要一些計算量，在重視效率的情況下，我們可以適當的改變調變網格的頻率，以增進程式的效率。

每次調變的對象是一整個階級的所有小片，如果我們對階級二的小片調變，意思是拿階級一小片的資料去尋找誤差較大的區域，然後進行切細動作產生階級二的小片。每次調變過後會產生全新的階級二的網格分佈，與調變之前階層二的網格分佈沒有絕對關連性。在新舊的網格重合的區域，則舊網格的資料取代。而沒有舊資料的區域，我們則用粗一層的資料插分。

調變網格有兩個重點，調變標準與插分方法，我們將在 2.4.1 節與 2.4.2 節說明。

2.4.1 調變標準設定

如何尋找誤差的來源，是網格調變的重點之一。一般來說，可以藉由相鄰格點物理量的相對差值來判斷。我們會依照不同的物理問題做特別的處理，如果模擬爆炸波(Sedov-Taylor Blast Wave)，我們可以簡單的利用流體密度或壓力做為判斷的依據。當兩個格點之間密度差大於某個標準(Criterion)，就判定這個區塊須要用更細密的網格處理，並對該小片做個標記。如果是渦流(Vortex)問題，我們可以用 $\left|\vec{\nabla} \times \vec{v}\right|$ 作為判斷的依據，或是用相鄰格點的速度方向所夾的角度來判斷是否處於渦流的中心位置。對於不同的模擬問題，有不同的判斷方式，一般來說，我們可以先分析誤差的來源會與哪個物理量變化有對應，再寫出相對應的程式碼去做判定，決定各個區域的網格粗細。

除了對格點上的物理量做分析外，也有另一種方法可以找出產生誤差的位

置，就是 Error Estimation[5]。其主要的概念，是利用粗與細網格計算結果的誤差來判斷。誤差大於設定的標準值時，就將網格切細；反之如果低於某個標準值，就將細網格移除。

為了增進程式的效能，判定時所作的計算量要盡量精簡，避免在判定時做太大的運算。目前判定函式是由中央處理器運算，相較於使用顯示卡運算的流體演化核心，運算能力差了十倍之多，太複雜的判斷，就會明顯地拖慢程式效率。如果遇到的判定複雜的問題，我們也可以使用高速的顯示卡進行判定的運算，改善程式效率。

被判定為誤差較大的區域，我們除了標記這些區域之外，會在這些區域旁，再加上幾個網格點的緩衝(Buffer)，如圖 2.4。圖 2.4 是個二維的例子，圖中是一個 8 × 8的小片，每個格子是小片中的網格，紅色區域是被判定為解析度不足的區域，綠色區域是我們加的緩衝，任何包含紅色或綠色區域的小片都必須被切細。這樣做的目的是為了避免在下次執行網格調變前，這些誤差大的訊息已經跑到了不夠細密的區域。

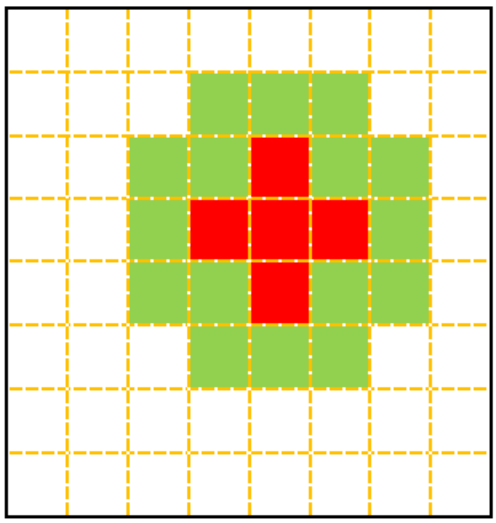
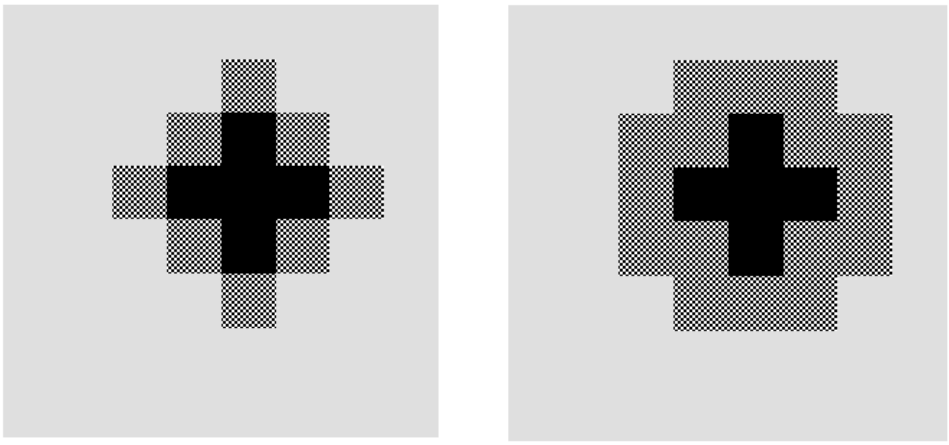


圖 2.4： 網格緩衝示意圖



(a) (b)

圖 2.5：適當網格分佈示意圖

在 AMR 的程式裡，我們常會要求粗細的網格分布必須滿足適當網狀結構 (Proper Nesting)。適當網狀結構的目的是，不希望粗細界面的解析度差距太大。在本程式裡，我們不容許階級差超過兩層的粗細界面。實作上的判別方式是，任何一個小片的父小片必須完整的擁有每個鄰小片。在三維裡，包含所有對角線方向上的鄰小片，總共有 26 個。除了階層為零或是與真實物理邊界有接觸的小片外，所有小片都必須要滿足這個條件。如圖 2.5a 與圖 2.5b，灰色區域是最粗網格的區域，方格狀區域是次細網格的區域，黑色區域則是最細網格所在區域，而圖 2.5a 是滿足適當網狀結構的例子，反之圖 2.5b 則因為最細區域與最粗區域在對角線方向上有接觸，所以不滿足適當網狀結構。

滿足適當網格結構的目的有兩個，第一是為了避免產生太多因為網格粗細差異太大的邊界而造成的計算誤差，因為適當網狀結構只允許粗細交界面階層只差一層的情況。第二個目的是減少粗細網格界面插分所造成的誤差與程式複雜度，因為所有小片的邊界，只有兩種可能，一個是與自己一樣粗細，另一個是與父小片同樣粗細。

為了滿足適當網狀結構，我們必須檢查所有小片的是否都有 26 個鄰小片，如果沒有，那麼就必須抹除之前需要切細的判定，不能讓沒有鄰小片的小片繼續往

下切細。這部分會造成程式的不理想，因為被判定為需要切細的區域並未被切細。不過在本程式的架構下，我們可以藉由判定標準和緩衝大小的改變來盡量避免這部分產生的計算誤差。

另外在 2.2 節討論到的流程裡，我們注意到在模擬中網格調變的順序先是細格點調變再來才是粗格點調變。有時候在細格點處判斷仍需要更加切細的區域，再粗格點會因為解析度不足的關係並未被判定，這時我們必須要特別為這些擁有更細結構的區塊加上標記，避免將這些細部資訊抹除。

2.4.2 網格插分

在尤拉方程式的流體模擬裡，讓守恆量維持守恆是一個非常重要的準則，所以在切細時，我們只能選用守恆插分。

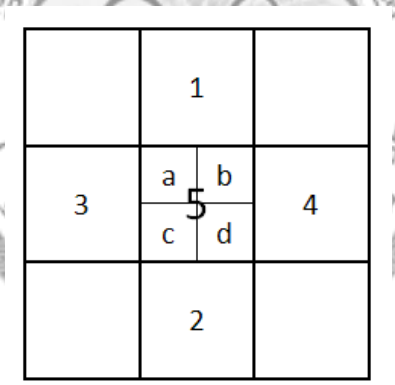


圖 2.6：網格插分示意圖

如圖 2.6，如果我們要對網格 5 插分出 abcd 四個格點，那我們選用的插分方式是

$$\begin{cases} \rho_a = \rho_5 + \frac{1}{4}[\ (\rho_1 - \rho_2) + (\rho_3 - \rho_4)] & (7) \\ \rho_b = \rho_5 + \frac{1}{4}[\ (\rho_1 - \rho_2) - (\rho_3 - \rho_4)] & (8) \\ \rho_c = \rho_5 + \frac{1}{4}[-(\rho_1 - \rho_2) + (\rho_3 - \rho_4)] & (9) \\ \rho_d = \rho_5 + \frac{1}{4}[-(\rho_1 - \rho_2) - (\rho_3 - \rho_4)]. & (10) \end{cases}$$

如此一來 $\rho_a + \rho_b + \rho_c + \rho_d = 4\rho_5$，滿足守恆量守恆，且運算量少，不影響程式效率。

2.5 模擬時距的決定

一次演化的時距(Time Step)，由 CFL condition 決定，其基本概念是訊息最大的傳遞速度是聲速加上流速，在一次演化裡，我們不容許任何訊息走超過一個格點，因為每次演化我們模擬的訊息傳遞也只會影響到隔壁的網格。於是對於某格點所能接受的最大時距是$\Delta t \times (V_{max} + C_s) \leq \Delta x$，其中$\Delta t$是演化時距，$V_{max}$是 V_x, V_y, V_z 三者取最大值，C_s 是該網格的聲速，可由方程式$C_s = \frac{\gamma P}{\rho}$求得，$\Delta x$是網格寬度。詳細的數學分析可參考 A Primer on Eulerian Computational Fluid Dynamics for Astrophysics [3]。

我們必須找到一個適當的時距，滿足每個格點允許的時距限制，由於在本程式的架構下，相鄰兩層的網格，細網格的Δt與Δx都恰好粗網格的一半。在程式裡，我們定義最細網格的寬度為單位長度，於是各階層網格的寬度為$\Delta x(lv) = 2^{ML-lv}$，各階層網格的時距為$\Delta t(lv) = 2^{ML-lv} \times \Delta t(ML)$，其中 lv 是各階層的階級，ML 是最細格點的階級，階級越大代表網格越細。而$\Delta t(ML)$可由圖 2.7 的虛擬碼求出。在實作上我們會對演化時距加個安全系數，因為在最粗格點演化一次(Global Time Step)的時間裡，細格點會使用相同的時距演化多次。

```
float determine_dt()
{
        dt_ML = "a large number";

        for(every ceils)
        {
           if( dt_ML > 1 / (Vmax + Cs) )
               dt_ML = 1 / (Vmax + Cs);
        }

        return dt_ML;
}
```

圖 2.7： 程式虛擬碼

```
void Intergration( current_level )
{
    for(  twice )
    {
        evolve(current_level);
        //Call the solver to advance current level

        if( finer level exist )
        {
            Intergrate(next_level);
        }

        fixup(current_level);

        refine(next_level);
    }
}
```

圖 2.8： 程式虛擬碼

2.6 演化

在圖 2.2 裡，Intergration 函式就是將流體演化到下一個時刻點的主要函式，也是本程式中流體模擬的核心。函式內的基本步驟已經在 2.2 節裡做過說明，在本小節裡，我們將更深入的介紹 Intergration 裡各個子函式。

2.6.1 演化簡介

圖 2.8 是 Intergration 的虛擬碼，其中 evolve 函式就是演化每一層網格的函式，fixup 函式就是將細格點算出來的值往上修正粗的格點，refine 函式就是進行每一層網格調變的函式。其中 refine 函式的重點已經在 2.4 節說明過了。

evolve 函式主要的內容，就是將各個小片的資料轉換成較適合 GPU 運算的簡單格式，並上傳到顯示卡的記憶體交由顯示卡運算。並且存下粗細邊界上的通量(flux)，留給 fixup 函式做向上修正。關於使用 GPU 運算的細節，我們會在 2.6.2 說明，fixup 函式我們會在 2.6.3 節說明。

2.6.2 顯示卡運算的相關細節

在第一章裡我們簡介了顯示卡擁有的強大運算能力，來自於使用多個 stream processors 作平行的運算。網格流體的模擬，每個網格點間的運算是獨立的，恰好是個適合於平行運算的程式架構。

如同 CPU 運算一樣，顯示卡也是從記憶體將資料讀進來，在 GPU 內做各種判斷與運算。不過由於顯示卡的計算能力太強大，如果記憶體讀取的效率不彰，將會非常嚴重的限制顯示卡的能力。在 CUDA 裡面，顯示卡的記憶體分為三種，分別為顯示卡的主記憶體(global memory)、共享記憶體(shared memory)、與暫存器(register)。在 CPU 裡也有類似的架構，分別為主記憶體(main memory)、快取(cache)、與暫存器。在顯示卡中讀取主記憶體的延遲(latency)是讀取共享記憶體的數百倍之多，約為 200~400 個時脈週期。所以要將顯示卡的計算能力完全發揮，

一定得要善用共享記憶體。

一張顯示卡內有多個串流處理器，在 CUDA 的架構底下，將 8 個串流處理器組成一個複合處理器(multiprocessor)，每個複合處理器擁有一個大小為 16 kb 的共享記憶體(shared memory)，提供給這 8 個 stream processors 作更快速的存取。不同的複合處理器是不能互相讀取其它複合處理器的共享記憶體。

在本程式裡，我們讓一個小片的資料用一個複合處理器進行演化，善用共享記憶體來儲存各個獨立小片的資料，讓顯示卡能夠更高速的運算而不會受到主記憶體讀取效率不彰的影響。而顯示卡除了負責各個小片的演化之外，不用處理任何複雜的資料結構與邏輯判斷。所以其它的運算，都交由 CPU 來負責，如此我們更可以善用顯示卡的優勢：單純直接的大量運算。

為了讓每個小片都是個獨立的運算，每個丟給顯示卡運算的小片，都必須擁有足夠的邊界條件。而我們使用的 TVD solver 在每一個邊界需要三格的邊界值(ghost zone)，所以我們先將每個小片與其所需要的邊界值複製到一塊新的記憶體上，才送入顯示卡演化。各小片的邊界值的取得方式，則看其是否有鄰小片存在。如果其鄰小片存在，邊界值就可直接從鄰小片複製，反之我們就需要對更粗一層的格點插分，得到所需要的邊界值。

我們需要另外注意的是，邊界值的處理除了再送進 GPU 前的準備需要花費額外的時間，在演化的過程中，它也是嚴重的影響整體效率。當我們演化 N^3 的小片，因為每個維度都有兩個邊界，所以每個維度我們需要額外 6 格邊界值，而每次只演化一個維度，如 X 方向，則在進行 Y 方向的演化時，X 方向的邊界值運算就可以省去，在 Z 方向上演化時，X,Y 方向的邊界值運算都可省去，故總共的運算為 $(N+6)^3 + (N+6)^2N + (N+6)N^2$。才能進行中間的小片演化。我們發現 N 越大，邊界值運算就越少，在 N=1 的情況下，邊界值的運算約是小片本身的 133 倍，是個極端資源浪費的例子。

所以小片的大小是個很重要的變數，太小則造成處理邊界值的資源浪費，太大則造成 AMR 粗細格點分佈的不理想，在第三章我們會對小片大小與效能做討

論。

為了增進程式效率，我們注意到產生小片的過程，都是由一個父小片產生八個相鄰的子小片(可參考圖 2.1)，為了節省邊界值的處理，我們將八個相鄰的小片視為一個更大的小片，一起送入顯示卡進行演化，增進程式效率。

由於顯示卡和 CPU 是兩個獨立的運算核心，CUDA 支援兩者分別進行獨立的運算。所以我們將大量的小片拆成多組小片，分批送入顯示卡。讓顯示卡進行演化運算的同時，CPU 可以處理下一批被送入的小片之邊界值，達成程式的最佳化。而在本程式裡，邊界值的處理所花費的時間正巧與 GPU 的進行演化所花費的時間相近，在這樣的功能底下，我們將程式的效率提高了兩倍。

2.6.3 向上修正

不同粗細的運算，由於誤差的關係，往往不盡相同，所以我們必須將細格點所得到的結果，對粗格點修正。所需要的修正主要分為兩種，一個是擁有子小片的小片，其格點的值，會完全被更細格點的平均所取代。另外是擁有粗細邊界的小片，粗小片與細小片間的流量必須要修正，讓守恆量維持守恆。

如果使用 Error Estimate(見 2.4.1 節)的方法調變網格粗細，則可在此時進行粗細格點計算誤差的比較，並標記誤差大的區域，在調變時產生更細的網格點。

關於粗細邊界流量的修正，我們必須另外建立一個新的記憶體空間，在每次演化時記錄粗細邊界的流量。而在顯示卡運算時，由於每個小片的邊界值在經過運算之後，其記憶體空間便空了出來，恰好可存放各邊界的流量，而不需要在顯示卡內另外宣告記憶體做儲存。

2.7 資料輸出

做電腦模擬最怕的就是跳電或是當機，每跑幾個小時做一次資料備份是一定不能省的。有時候我們也會做一些程式的變動，將模擬重跑，卻也不希望真的是重頭跑起，這時候有中途時刻點的資料備份，就會省時許多。

在本程式的架構底下，擁有細格點的區域，其粗格點的變數資訊可由細格點產生。在資料輸出時，為了減少輸出檔案的大小，我們只會將各個區域的最細小片儲存起來。

另外在第一章提到，本程式有因為時間插分的需求，所以會儲存兩個不同時間點的資料。不過在所有格點皆最粗格點演化同步時，前一刻的資訊就不會在使用到，故我們選在此時進行資料的輸出，可節省一半的輸出檔案大小。

3. 程式效能分析與測試

在第三章，我們測試了 Sod shock tube problem 與 Sedov-Taylor blast wave 兩個常見的流體測試。我們的效能對照所採用的 CPU 與 GPU 分別是 AMD Athlon™ X2 Dual-Core 3800+與 NVIDIA Geforce 8800 GTX。在測試當中，我們比較的是一張 8800GTX 的顯示卡與 AMD 3800+單一核心的運算效率。其理論運算效能分別為 8 GFLOPs 與 345.6 GFLOPs。

3.1 震波管

Shock tube 是個流體模擬的經典測試之一，絕大部分的流體模擬程式完成時，都必須先通過 shock tube 的考驗。Shock tube 的問題是兩個狀態不同的流體間的不連續面的演化，其中最有名的就是 Sod Problem[6]。因為 shock tube 有解析解的存在，可清楚的了解模擬的結果與實際物理之間的差異。

我們的初始條件是就是 Sod shock tube problem 的初始條件，各個變量的初始值：$\begin{cases} (\rho, u, e) = (1.0, 0, 1.5) \ , x \leq 0 \\ (\rho, u, e) = (0.125, 0, 0.15) \ , x > 0 \end{cases}$。

由於本程式是三維的流體模擬，雖然 shock tube 是個一維的測試，但程式實際上還是做三維的運算。在此我們測試最細格點為 256^3 的網格點，首先我們把我們的模擬結果與理論解相比，如圖 3.1~3.3。我們發現我們所使用的 Solver 可以清楚的描述演化出來的四層不同狀態的流體。

在與 CPU 的效能比較之前，為了確定 CPU 與 GPU 所做的是相同的運算，我們先比較兩者的模擬結果，我們發現 GPU 與 CPU 模擬結果幾乎完全相同，平均的相對誤差約在 10^{-7} 左右，誤差分佈如圖 3.4，基本上已經達到單精確浮點數的精確度。

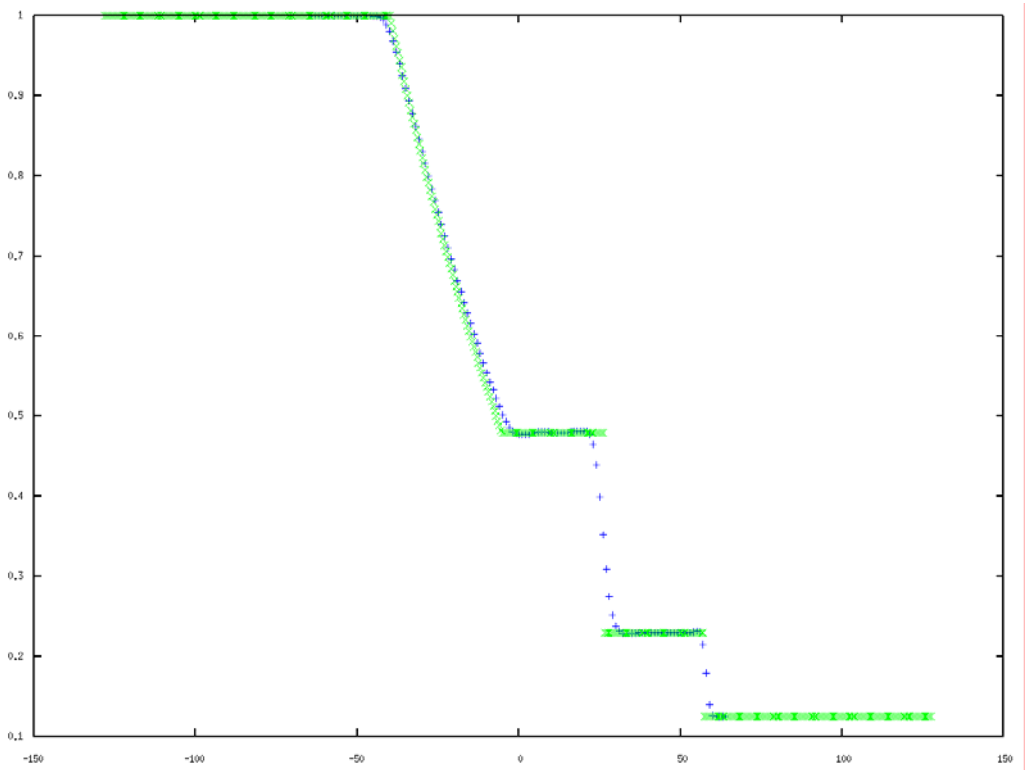


圖 3.1：Shock Tube 密度分佈。縱軸是密度，橫軸是空間座標。

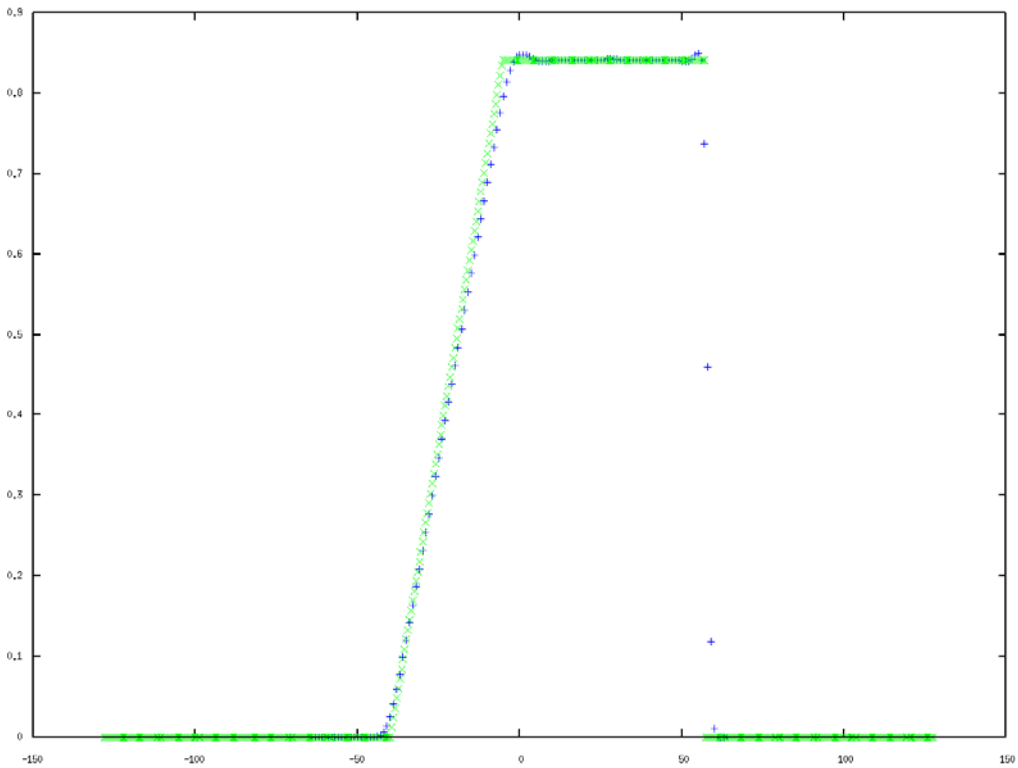


圖 3.2：Shock Tube 速度分佈。縱軸是速度，橫軸是空間座標。

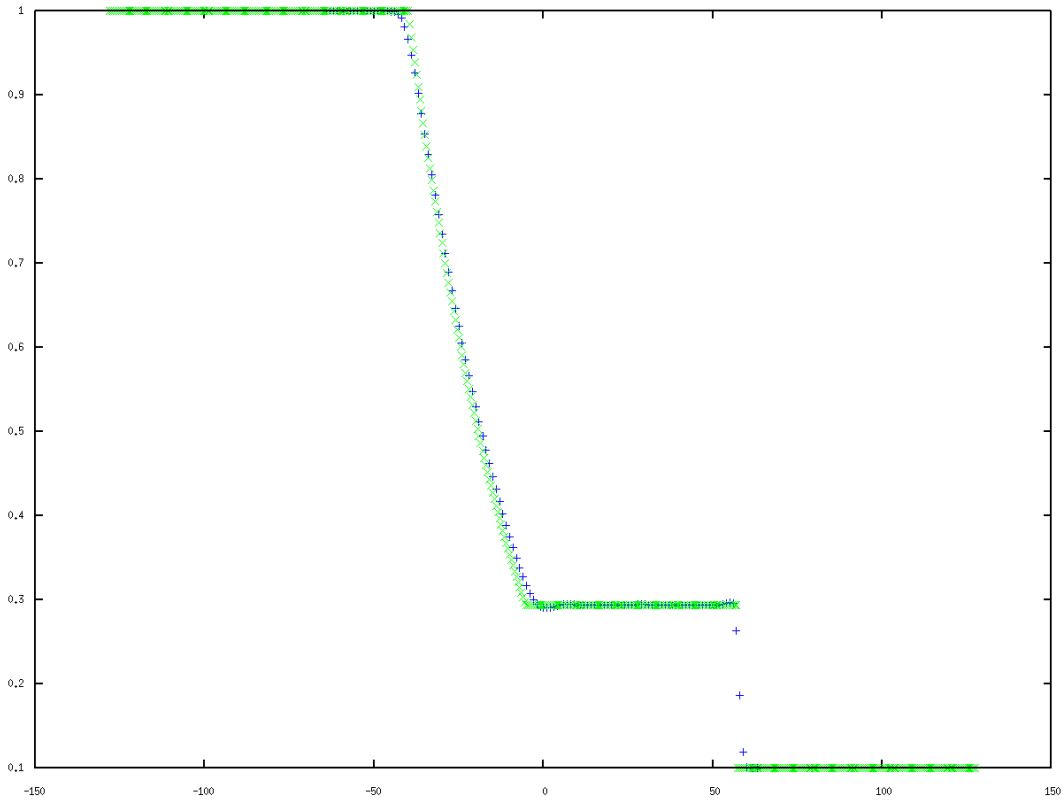


圖 3.3：Shock Tube 壓力分佈。縱軸是壓力，橫軸是空間座標。

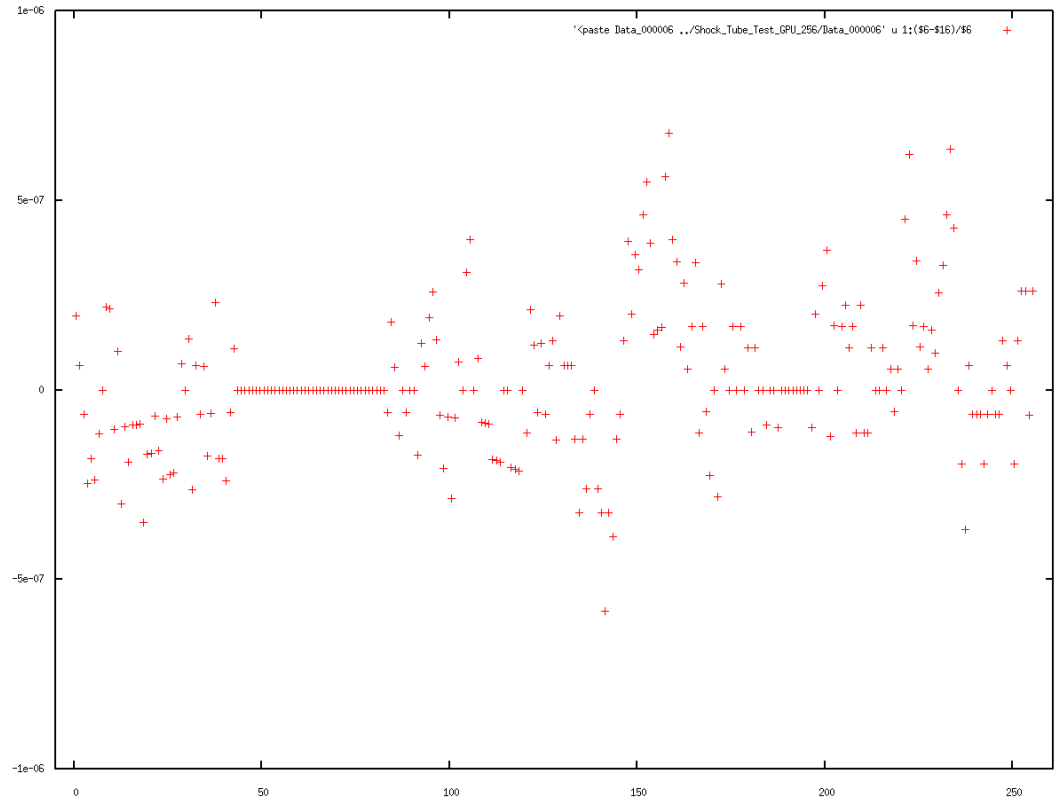


圖 3.4：震波管之計算相對誤差分佈圖。縱軸是相對誤差，橫軸是空間座標。

Patch size	4^3	8^3	16^3	32^3
GPU without AMR	375.13	212.80	371.36	1248.92
CPU without AMR	3869.45	2420.87	1979.60	1696.64
GPU with AMR	275.90	222.12	NA	NA
CPU with AMR	2488.03	1997.32	NA	NA

表 3.1：Shock Tube 運算時間表

表 3.1 是 CPU 與 GPU 在模擬 Sod problem 到 t = 40 的計算時間統計表。我們發現 GPU 在小片大小為 32^3 的效能低落，這是由於 GPU 的共享記憶體大小有限，在不改寫程式碼的架構底下，小片大小與共享記憶體使用率有一些關連性，在 32^3 的小片大小底下，共享記憶體的使用率很低，以致於程式效能低落。然而在程式設計就預期到小片大小太大會造成 AMR 的資源浪費，所以對於大小太大的小片，如 16^3、32^3 等大小，並沒有做更進一步的最佳化。而在小片大小為 4^3 時，由於邊界值的大小與小片大小的比例太高，以致於邊界值的處理費時甚鉅(見 2.6.2 節)，結果效能低於 8^3 的小片大小。在 CPU 方面，我們發現小片越大費時越少，這就是 2.6.2 節所說的邊界值的處理與小片大小的關係。

在 AMR 方面，由於本測試是一維的，所以只有一個維度會有不同的粗細網格分佈，這代表著能用粗網格運算來加速的區域變得更少，到了演化末期，各個不連續的界面分佈了全空間，幾乎全部的區域都切細成為最細網格。在 CPU 方面，由於演化部分運算量仍占大部分，少幾個最細網格點的運算，仍有明顯的加速。GPU 方面，由於 AMR 多次來的資料結構處理，都是由效能較低的 CPU 負責，多出來的時間花費大於 AMR 省下的時間，所以程式執行的效率反而變得更差。

3.2 爆炸波

Sedov-Taylor 爆炸波是最有名的流體測試之一，其模擬的是一股能量突然在空間中的一點釋放出來，向外釋放的衝擊波，可類比於炸彈的爆破或是超新星爆炸。

本測試的初始條件如下，

$$\begin{cases}(\rho,u,e)=(1.0,0,10^{10}) & ,|\vec{r}|=0\\ (\rho,u,e)=(1.0,0,10^{-20}) & ,|\vec{r}|\neq 0\end{cases}$$

。$|\vec{r}|$代表與爆炸中心的距離。

Sedov-Tayler 爆炸波解析解的來源是參考文獻[7]。我們此次的模擬使用最高解析度為 512^3 的網格點，圖 3.5~3.7 是理論解與模擬結果的對照圖，分別是密度、速度、與壓力對與爆炸中心距離所作的圖。由圖可知，本程式雖然是使用直角座標描述空間的流體，但是對於球狀分佈的爆炸波依然描述的十分恰當。

我們也針對爆炸波的模擬做不同小片大小的測試，得到表 3.2。由於均勻網格點的模擬需要花費極大量的運算時間，我們只提供使用 AMR 的效能對照表。我們發現不論使用 CPU 或是 GPU 運算，小片大小越大越費時，原因就是當小片越大時，原本不需要精細網格就可以模擬的區域被切細了，這些運算資源的浪費就顯現在運算時間上。圖 3.8 是此次模擬空間網格點的分佈圖，我們清楚的看到震波波前附近的格點切細到最細緻的網格，而其他大部分的平坦區域，我們使用粗網格模擬，達成 AMR 的理想。

與 shock tube 的效能分析相比，我們發現到在不同問題底下，提供最佳效能的小片並不相同。在精細結構越集中的問題裡，小片稍小會是個比較適合的選擇，反之在大多數區域皆會被切細的問題裡，則會因為邊界值的處理造成效能不彰，則我們可以選用稍大的小片。在未知結果的模擬底下，8^3 的小片大小可能是比較適當的平衡點。

我們也發現，不論是在爆炸波或是震波管的模擬，在最佳化的設定底下 GPU 的效能都約是 CPU 的 10 倍，這代表著我們已經發揮了 GPU 的優勢，順利的提高流體模擬程式的效能。

Patch size	4^3	8^3	16^3	32^3
GPU with AMR	1017.50	1254.33	3066.61	23650.44
CPU with AMR	8639.71	10874.27	22561.08	58182.21

表 3.2： Sedov-Taylor 爆炸波運算時間表

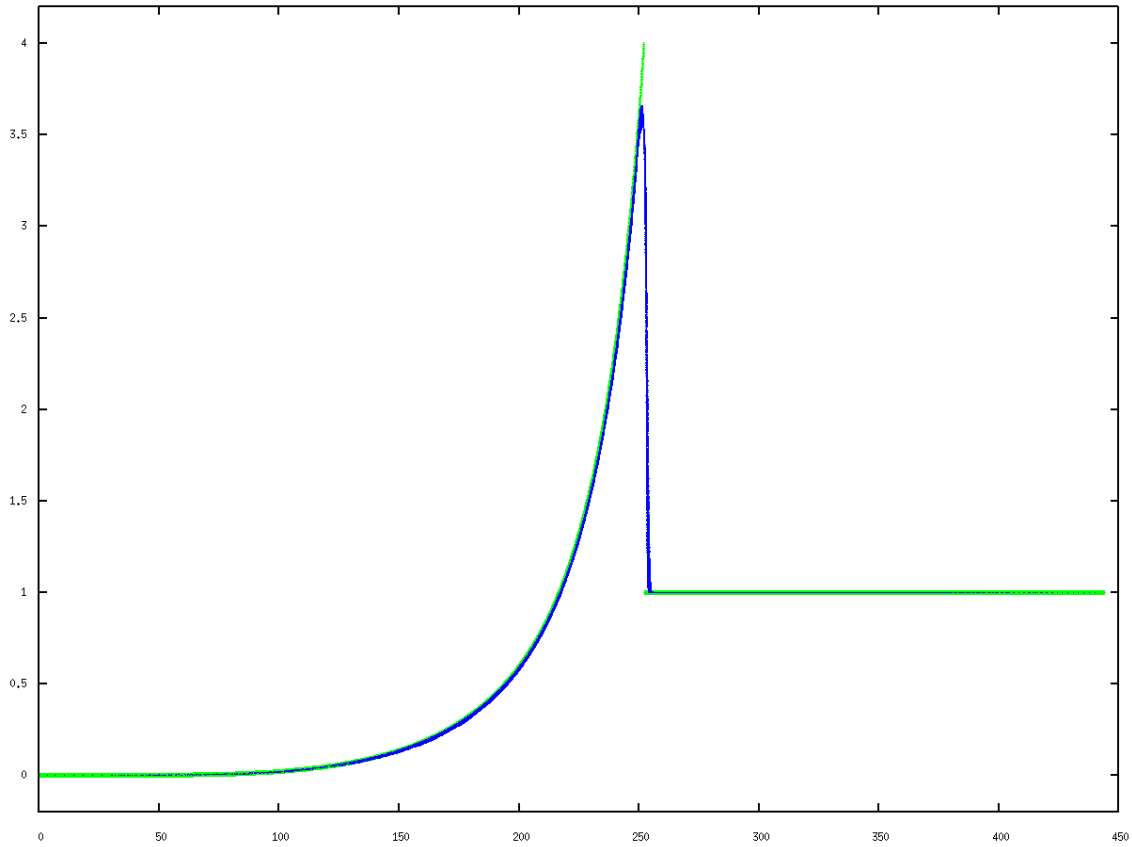


圖 3.5： Sedov-Taylor 爆炸波的密度分佈。縱軸是密度，橫軸是半徑。

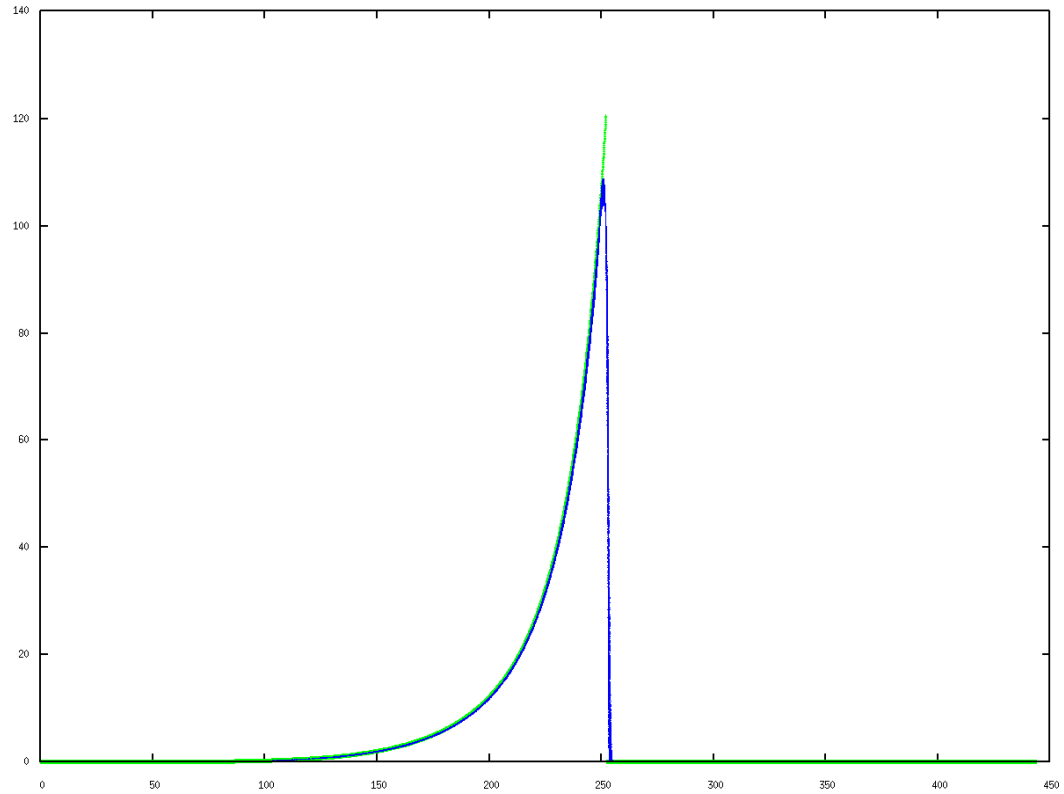


圖 3.5： Sedov-Taylor 爆炸波的動量密度分佈。縱軸是動量密度，橫軸是半徑。

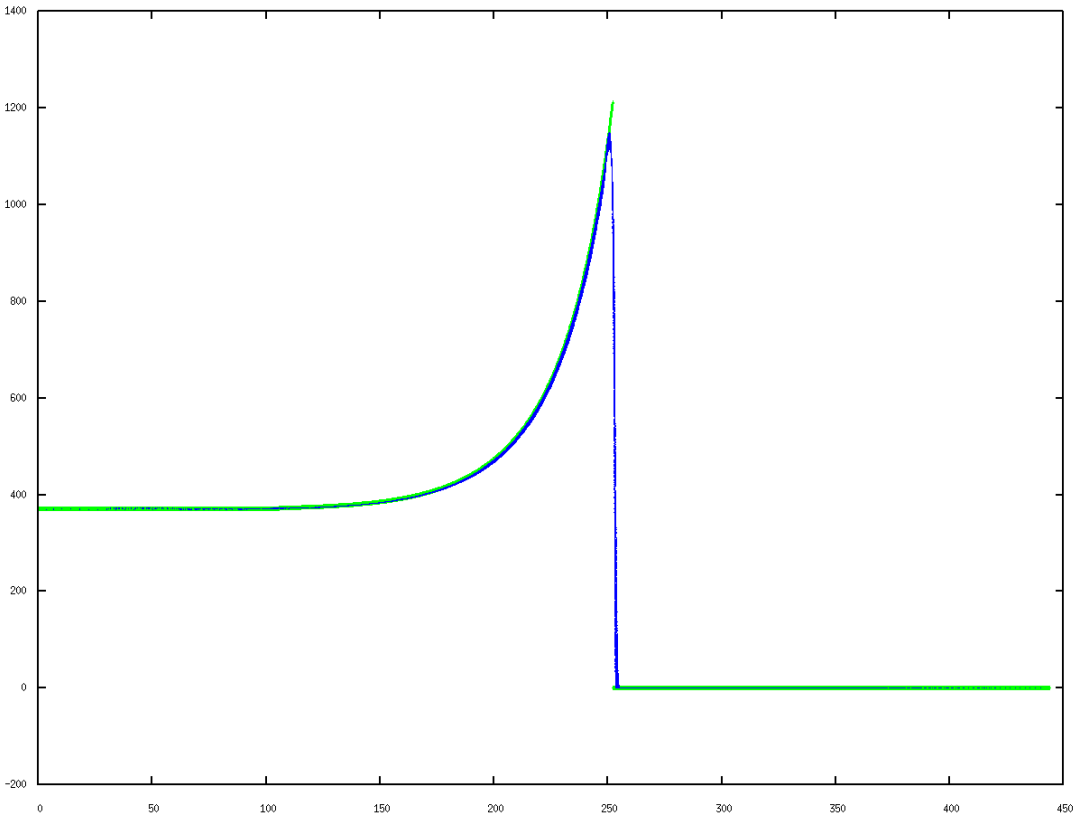


圖 3.5： Sedov-Taylor 爆炸波的壓力分佈。縱軸是壓力，橫軸是半徑。

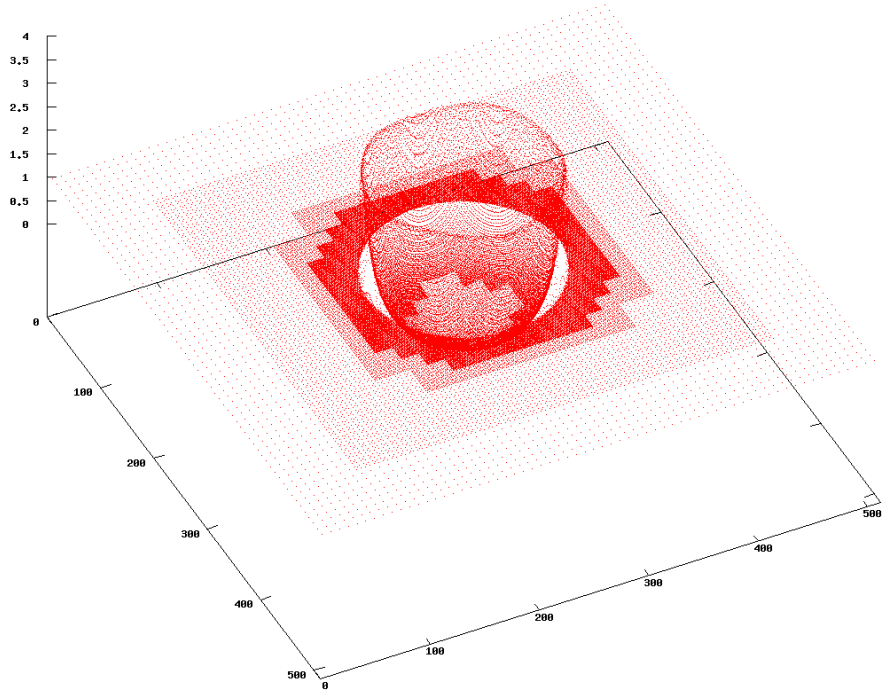


圖 3.8：爆炸波之空間網格分佈圖。Z 軸是密度，X,Y 分別是空間座標。

4. 紊流分析

4.1 紊流能量頻譜

紊流(Turbulence)是個少數沒有完美理論的古典問題，物理學家們不斷地試圖用各種方法分析紊流，從雜亂的流場裡找出其中不為人知的道理。西元 1941 年，Kolmogorov 發表了著名的能量頻譜(energy spectrum)[8]

$$E(k) \propto k^{-\frac{5}{3}}, \qquad (11)$$

其後的數年間，如雨後春筍般的實驗發表認同這樣的能量頻譜。之後 Kraichnan 將這個三維的能量頻譜擴展到二維[9]。近期由於科學計算能力大幅成長，以往無法達到的高解析模擬，也在近二十年來有大量模擬成果被發表。

首先我們稍微簡介 Kolmogorov 能量頻譜的來由。在紊流裡有大大小小的混亂結構，一般常見的是一股巨大地擾動在流體裡形成大尺度的速度場分佈，然後慢慢小尺度結構發展出來，在大尺度的流場裡出現各種尺寸的混亂渦流，直到整個流場的能量慢慢的消散，流體越流越慢，混亂結構慢慢消失，變成了均勻流場。如果我們持續在大尺度結構對系統擾動，那麼我們會發現這些混亂的渦流似乎一直在改變，但各種尺寸的結構似乎依然存在，整個流場看似快速地改變著，實際上不論哪一種尺寸的結構，總是在不同地點消失又出現。我們也會發現大尺度的流場似乎並不影響小尺度的結構，只是將小尺度的結構帶著走。而小尺度結構的產生和消滅，總是周圍相似尺度的結構造成的。於是 Kolmogorov 假設，大尺度結構不會直接造成小尺度結構的變化，總能量的消散是由最小尺度造成的，大尺度能量的損失僅僅是傳遞給小一點的結構，一路傳到黏滯力對流體的影響相對於慣性力來說不可忽略的小尺寸。在外界施加於大尺度流場的功率不變的情況下，在整個流場中各個單一尺寸的總能量將不會隨時改變。在這樣幾個的前提之下我們可以得到一個小結論，就是不論各種尺寸的流場結構向下傳遞的能量都會一樣，這樣才會造成每一個尺寸所帶有的能量不隨時改變，否則必定有某種尺寸的流場

發展起來。換句話說，在這樣的流場底下，不論尺寸大小，都有一樣的相似模型。於是 Kolmogorov 藉由以上幾個假設在高雷諾數的不可壓縮流體底下，經過統計力學的推導，得到了$-\frac{5}{3}$的能量頻譜。

4.2 模擬

4.1.1 初始條件

其他研究團隊用來模擬這種紊流問題的方法，常是在流體方程式裡加入了黏滯力的效應，而本程式試圖藉由數值誤差的能量消散方法來產生相同的結果。本程式使用的流體演化核心，嚴格要求總變化衰減(TVD)，其方法是強迫數值插分的誤差一定只會造成總變化衰減。而數值插分誤差大的原因，總是因為相鄰的格點物理量變化很大。而相鄰格點流場變化大的區域，通常就是黏滯力大的區域。

本次模擬的起始條件是

$$\begin{cases} \rho = 1, \\ \rho v_x = \sin\left(\frac{2\pi y}{L}\right) \\ \rho v_y = \text{random number of order of } 10^{-4}. \\ \rho v_z = 0; \\ e = 3 + \frac{1}{2}\rho v_x^2 \end{cases}$$

這裡的 L 是模擬空間的長度。

Kolmogorov 是在不可壓縮流體的前題下推導出$-\frac{5}{3}$的能量頻譜，而本程式並沒辦法模擬不可壓縮流體。為了得到相似的結果，我們將壓力提高，希望在模擬途中密度的改變不會太劇烈。但我們並不能將壓力設得太高，原因是我們模擬時距跟聲速有很大的關係，聲速太高，模擬的演化就會變得很慢。在 Y 方向上的速度加入亂數的目的，是希望這個系統的不穩定性快速成長，產生我們希望看到的紊流。

由於這個系統並沒有外加的能量，所有的能量來自於初始條件中大尺度的切變流。這代表著在時間的演化下，總動能會逐漸衰減，各個尺度下的能量也將隨

時改變，但是我們預期依然能在能量頻譜中看到一些冪次的關係。

4.1.2 分析

首先我們第一個想要問的，就是這樣的模擬到底可壓縮性有多高？原來的 Kolmogorov 能量頻譜來自於不可壓縮流體，如果我們此次模擬的可壓縮性不可忽略，那就幾乎不能期望我們的模擬結果具有 Kolmogorov 的能量頻譜。

我們將可將流場分為可壓縮部(compressible)與螺旋部(solenoidal)

$$\vec{V} = \vec{V}^c + \vec{V}^s, \qquad (12)$$

$$\text{其中} \vec{\nabla} \cdot \vec{V}^s = 0, \vec{\nabla} \times \vec{V}^c = \vec{0}$$

我們可對這兩個部分分別做能量頻譜分析，與總能量頻譜相比，我們得到圖 4.1。我們可清楚的看到螺旋部的能量頻譜幾乎與總能量頻譜貼在一起，而可壓縮部的能量頻譜很明顯小了好幾個數量級，這代表著我們能量頻譜主要由螺旋部主宰，其可壓縮性幾可忽略。我們也做了密度的頻譜分析，如圖 4.2。

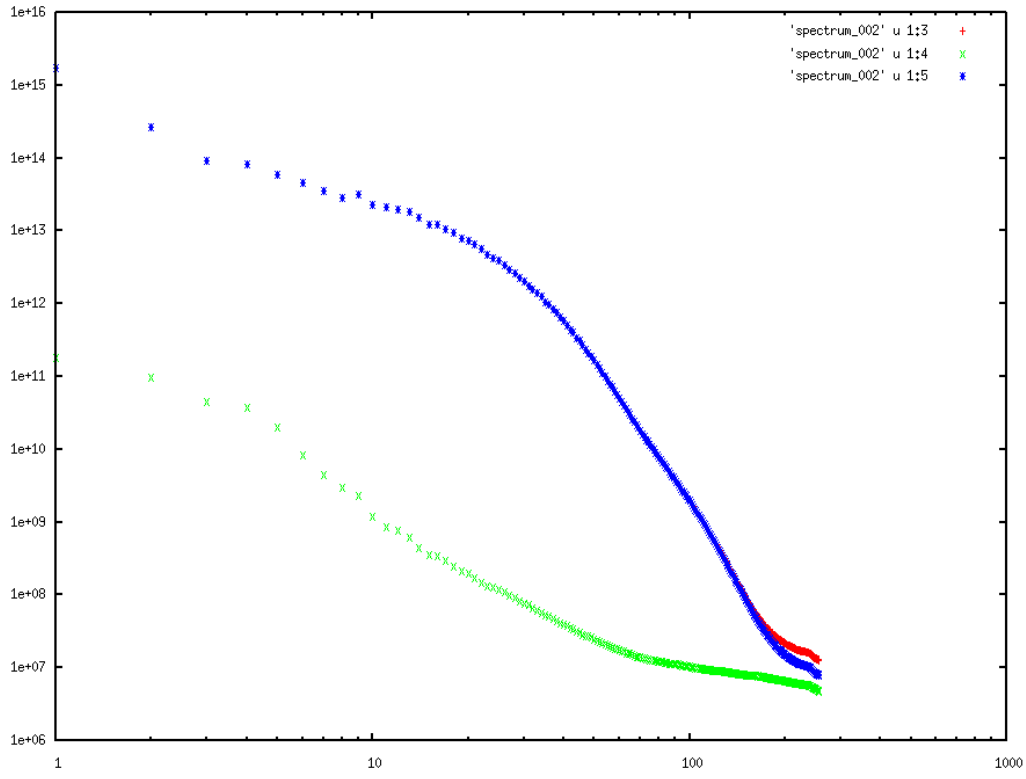


圖 4.1：能量頻譜。縱軸是 E(k)，橫軸是 k。圖中紅線是 E、綠線是 E^c、藍線是 E^s。

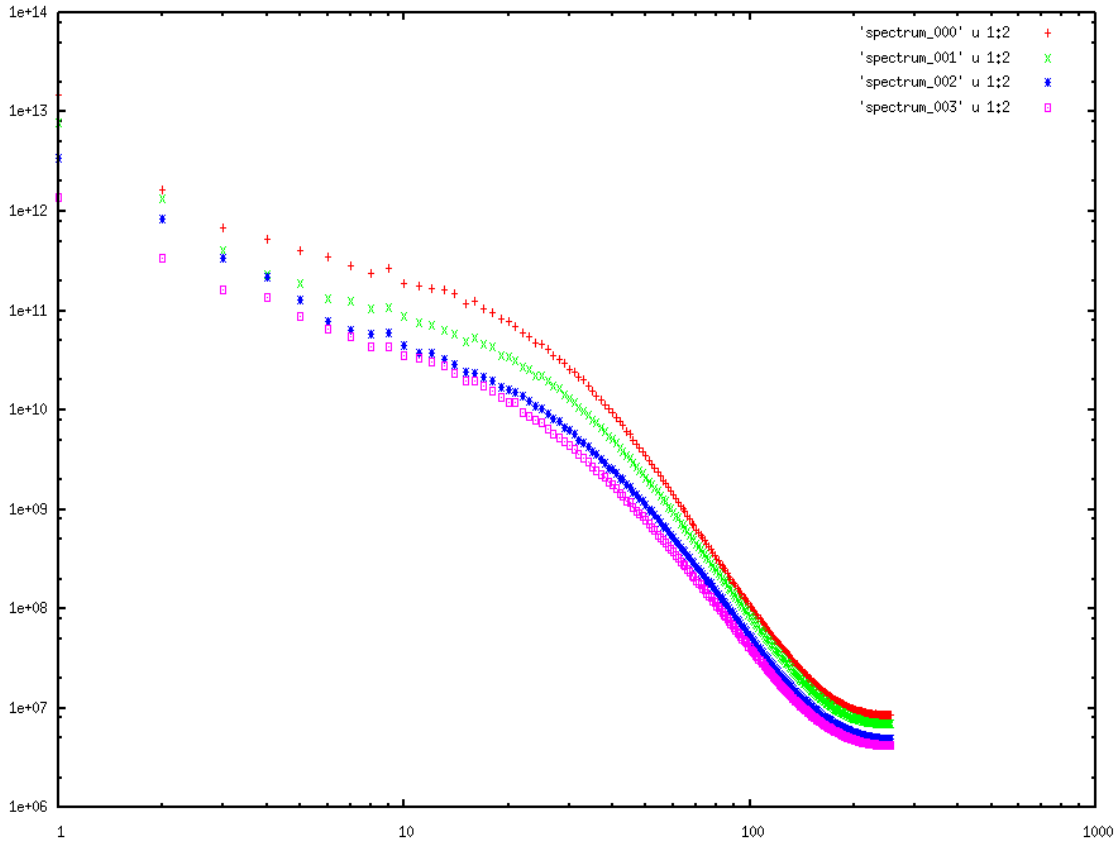


圖 4.2：多時間點的密度頻譜。縱軸是Δρ(k)，橫軸是 k，越下方的代表越晚期。

除此之外，我們的初始條件似乎紊流的基本假設之一「等向性(isotropy)」有點出入。我們預期這樣的切變流不穩定，應該要產生紊流。但是這樣的初始條件產生的紊流是否真的具有等向性呢？於是我們對密度與速度分別對三個方向作頻譜分析，如圖 4.3~4.5。我們發現在此次模擬中，等向性非常的完美。

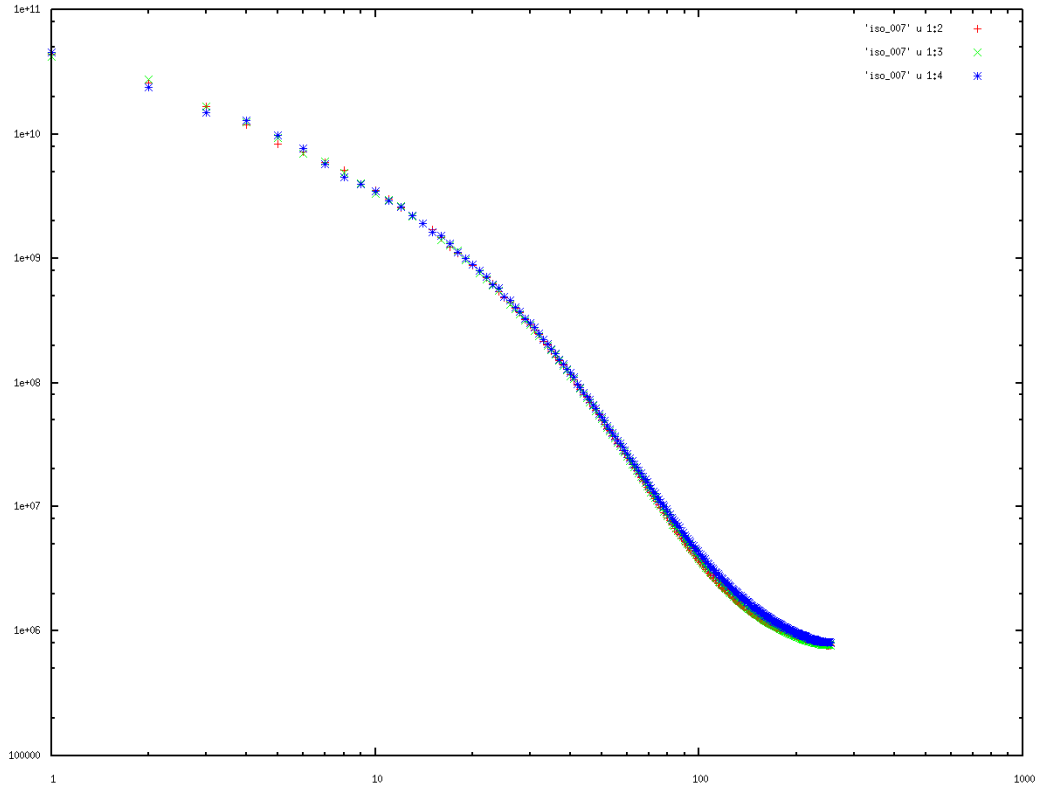


圖 4.3：密度在各方向上的頻譜。縱軸是Δρ(k)，橫軸是 k。紅線是 X 方向、綠線是 Y 方向、藍線是 Z 方向。

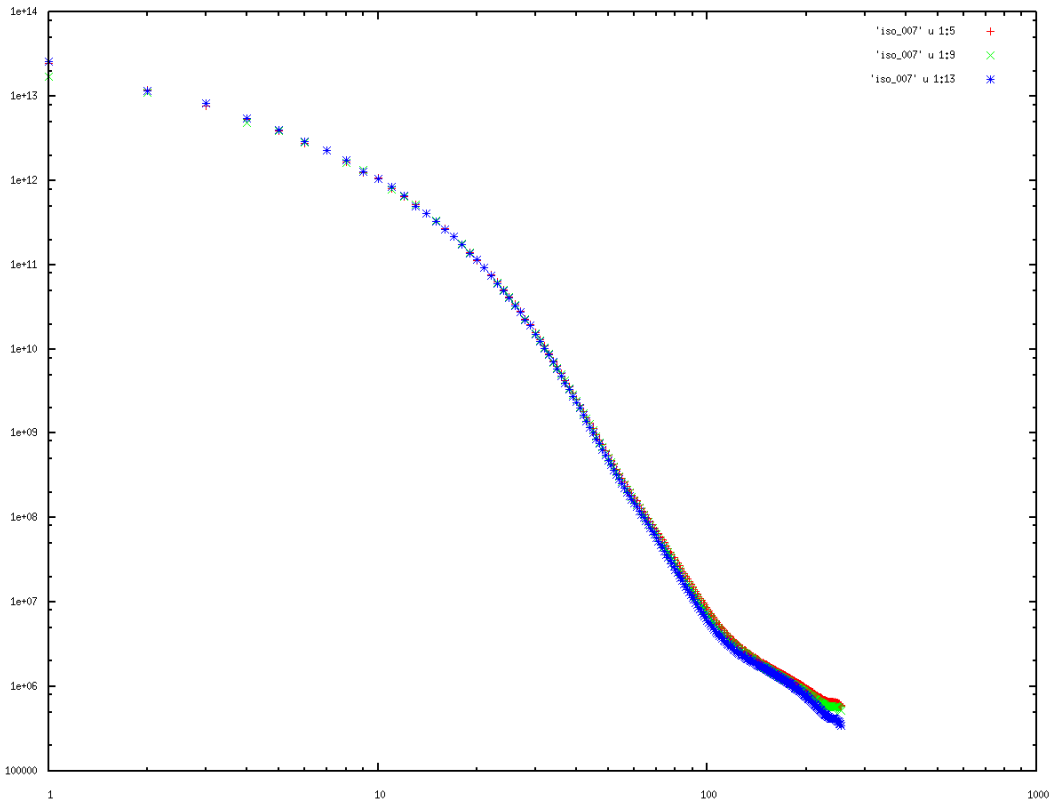


圖 4.4：速度在各方向上的頻譜(1)。縱軸是 E(k)，橫軸是 k。
這是 Vxx、Vyy、與 Vzz。

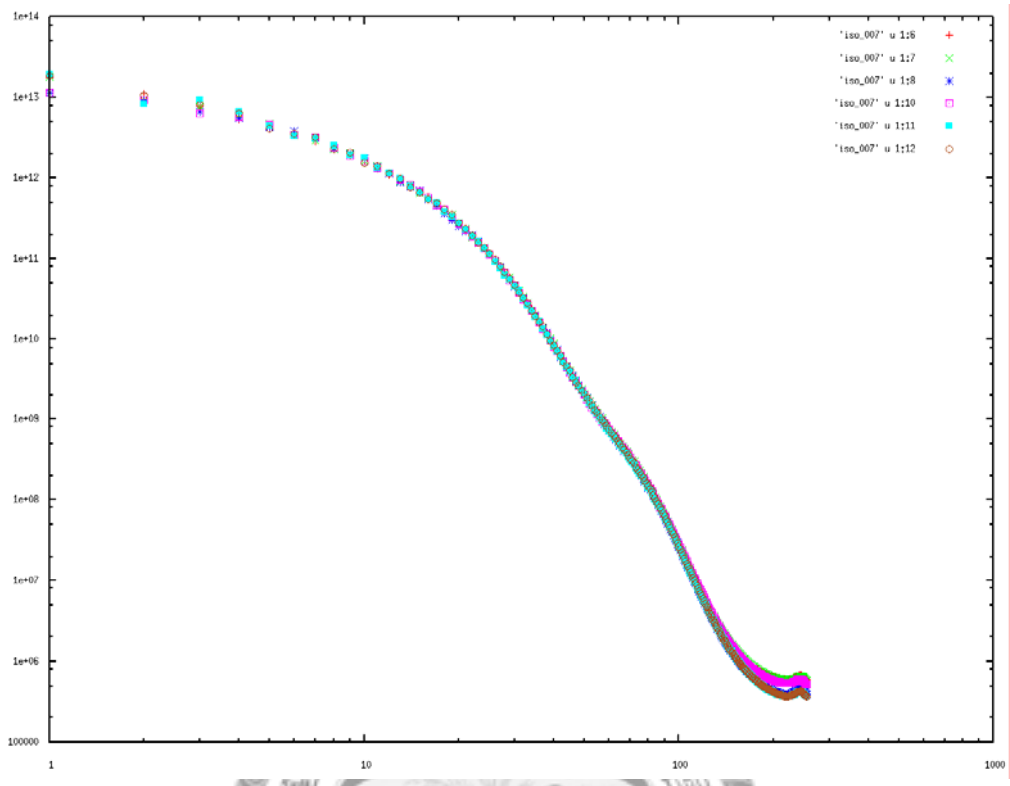


圖 4.5：速度在各方向上的頻譜(2)。縱軸是 E(k)，橫軸是 k。
這是 Vxy, Vxz, Vyx, Vyz, Vzx, Vzy。

最後我們比較各個不同時刻點的能量頻譜，如圖 4.6，可以看到能量逐漸消散，但在慣性區域(inertial range)裡-1 的冪次看起來穩定不變，這是個使用 AMR 的模擬。

我們希望確認使用 AMR 與均勻格點的模擬的差異，於是我們也從之前的模擬中，抓一個尚未產生渦流的早期輸出檔做為起始條件。圖 4.7 是多個時刻點的能量頻譜，我們發現與 AMR 之模擬結果十分相符。不過值得一提的是，由於我們的黏滯力來自於數值插分誤差，在粗網格的模擬與細網格的模擬形成的時間點並不相同，原因是粗網格的誤差較大，產生紊流的時間較早。

這樣的頻譜冪次與 Kolmogorov 提出的冪次的差異，可能來自於我們使用的流體演化核心是一個理想流體的模擬，並不是一個不可壓縮的流體模擬，且我們的黏滯力效應產生自數值誤差，而非手動放進去的已知黏滯係數。如果希望使用特定的黏滯係數的模擬演化核心，我們只須改寫演化核心部分的程式碼即可，對於

我們的 AMR 架構絲毫不會有影響。

David H. Porter & Paul R. Woodward 所發的期刊中[10]，其所作的理想流體模擬也得到與我們相同的-1 冪次關係，如圖 4.8。或許在理想流體裡，紊流的能量頻譜就是-1 冪次，這目前還是一個沒有定論的問題。

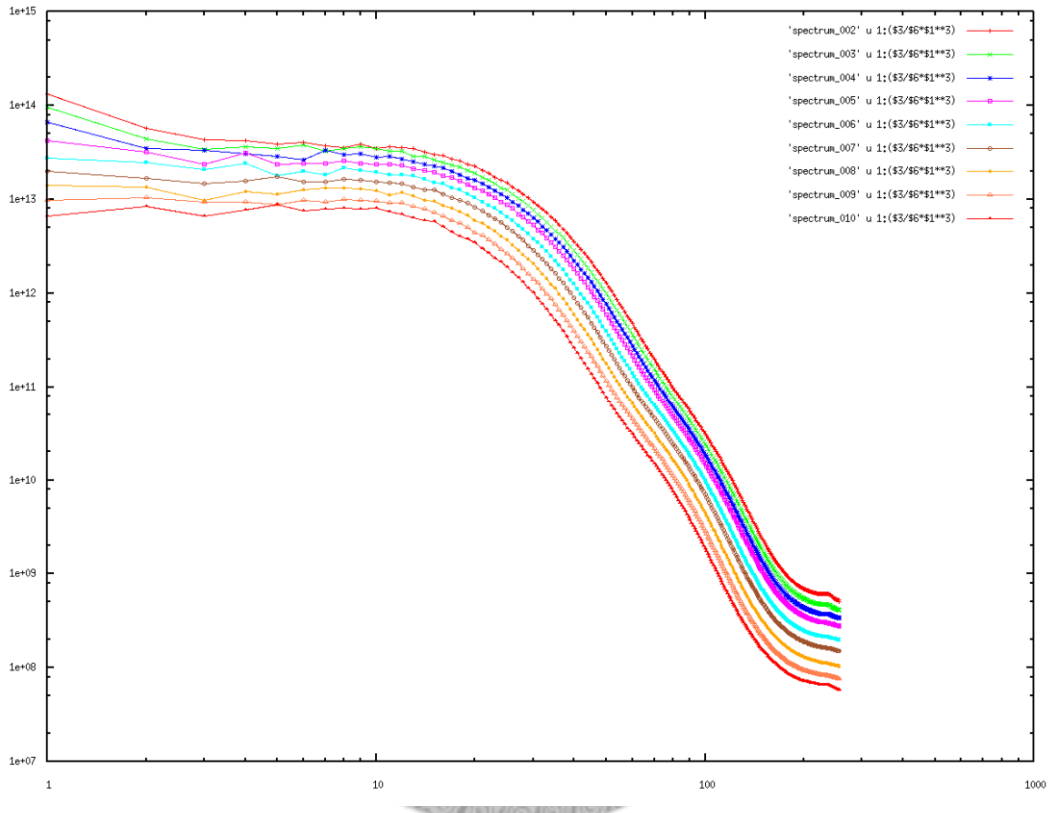


圖 4.6：使用 AMR 模擬的能量頻譜。縱軸是 E(k)*k，橫軸是 k，越下方時間越晚。

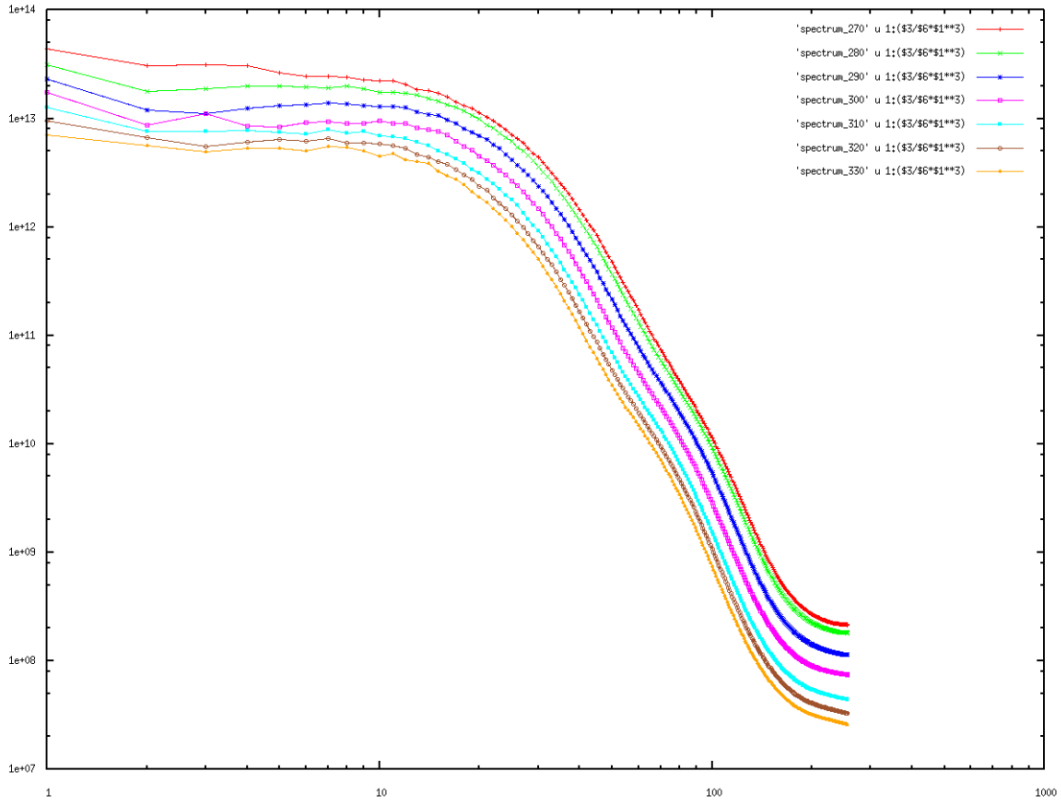


圖 4.7：使用均勻網格的能量頻譜。縱軸是 E(k)*k，橫軸是 k，越下方時間越晚。

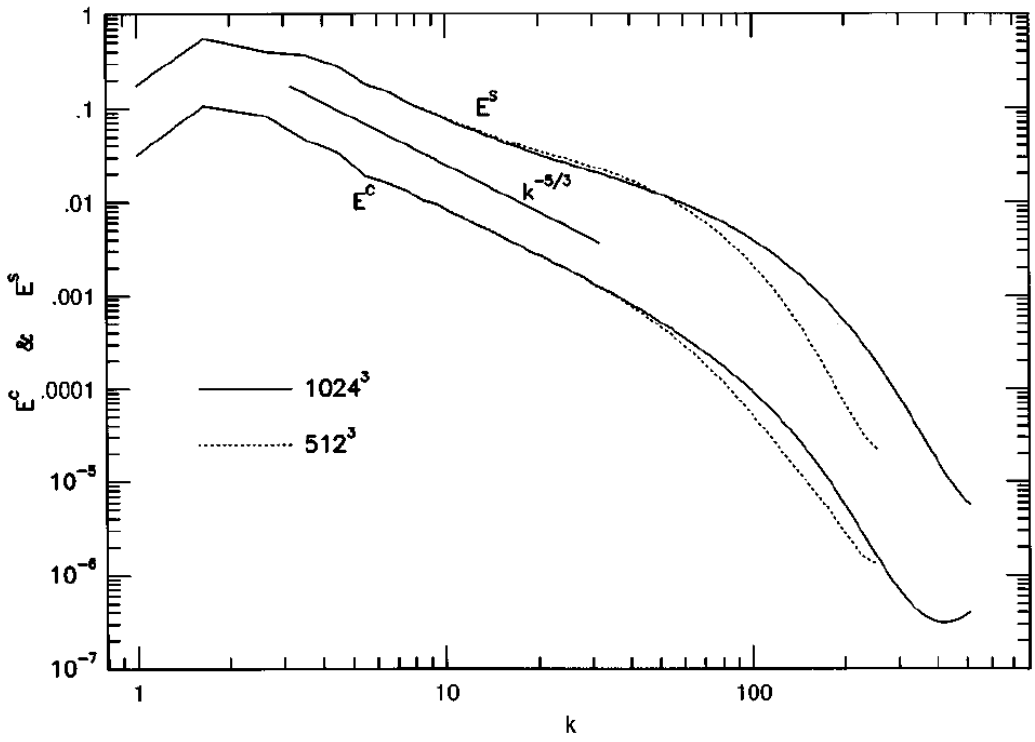


圖 4.8：D. H. Porter, P. R. Woodward 的能量頻譜。

5. 結論

本程式經過測試可正確的描述震波與紊流系統，並提供比以往模擬程式高出十倍的程式效能，帶領未來天文與其他領域的流體模擬進入更高速的領域，將可在有限時間內提供更高解析度的模擬。AMR 的程式架構將平坦與較無意義的區域用足夠的解析度演化，使絕大部分的運算資源得以發揮在讓人感興趣的區域。配合上超高速顯示卡運算核心，使得以往只能望之興嘆的物理問題出現了轉機。

未來本程式將進一步的加入更多功能，如磁流體、重力效應、與粒子模擬，能真實的處理天文問題，如星系形成、恆心形成、星際物質模擬，星系際物質模擬等許多以往因解析度不夠而放棄的物理問題。

本程式的平行化目前已經著手編寫，將於幾個月內完成，未來將可使用數台平價的個人電腦，提供可達地球前百大的計算能力的個人電腦平行運算叢集。

目前顯示卡的運算仍有許多缺點與不便，雖然現在 CUDA 已經提供了方便的介面，但要能夠完全的發揮顯示卡的運算能力，一定要是個可以被平行化的大量運算，否則顯示卡的效能很有可能會低於中央處理器。另外顯示卡目前最大的缺點就是僅提供單精確浮點數的運算。在科學計算的領域，單精確浮點數經常無法精確地描述物理現象。不過以目前通用顯示卡運算的突飛猛進的態勢，雙精確浮點數的支援是勢在必行的。

參考文獻：

[1] M.J. Berger, J. Oliger, Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations, Journal of Computational Physics, Volume 53, 484-512, 1984.

[2] http://www.nvidia.com/object/cuda_home.htm

[3] Hy Trac, Ue-Li Pen, A Primer on Eulerian Computational Fluid Dynamics for Astrophysics, The Astronomical Society of the Pacific, Volume 115, 303-321, 2003.

[4] P. MacNeice, K. M. Olson, C. Mobarry, R. deFainchtein, and C. Packer, PARAMESH : A Parallel Adaptive Mesh Refinement Community Toolkit, Computer Physics Communications, Volume 126, 330-354, 2000.

[5] J. Oliger, X. Zhu, Stability and Error Estimation for Component Adaptive Grid Methods, Applied Numerical Mathematics, Volume 20, 407-426, 1996.

[6] G. Sod, Survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, Journal of Computational Physics, Volume 26, Issue 4, 1-31, 1978.

[7] A. Cheng, Stability of the Sedov-Taylor Blast Wave Solutions, Astrophysical Journal, Part 1, Volume. 227, 955-957, 1979.

[8] A. Kolmogorov, The Local Structure of the Turbulence in Incompressible Viscous Fluid for Very Large Reynolds Numbers, Dokl. Akad. Nauk SSSR, Volume 30, 301-306, 1941.

[9] R. H. Kraichnan, Inertial-Range Transfer in Two- and Three-Dimensional Turbulence, Journal of Fluid Mechanics, Volume 47, 525-535, 1971.

[10] D. H. Porter, P. R. Woodward, Inertial Range Structures in Decaying Compressible Turbulent Flows, Physics of Fluids, Volume 10, 237-244, 1998.