— 平行程式設計 HW2 報告 —

$$\int_{-\infty}^{\infty} \frac{\sin x}{x}\, dx$$

1. 實驗目的與背景

- **目的**：將多種平行方法放在同一問題、同一硬體、同一測試流程下比較，不僅能揭開效能迷思、增進理論與實務連結，更能為後續優化、移植與學術溝通奠定穩固基礎。對學生而言，這種「同題多解」的練習，是理解平行計算本質與工程取捨的最佳途徑；對研究與產業，也能提供選擇最合適平行策略的實證依據。
- **目標**：探討在同一硬體（4 核心）上，四種平行方法（MPI、$POSIX\ Threads$、$OpenMP$）與序列程式在 **計算效率** 與 **數值正確性** 的差異。
- **方法**：以 Midpoint Rule 近似積分，將區間 $[-100, 100]$ 均分為 $N = 10^8$ 子區間。
- **理論值**：$\pi \approx 3.14159$；截斷及離散化誤差容許值約 0.55 %。

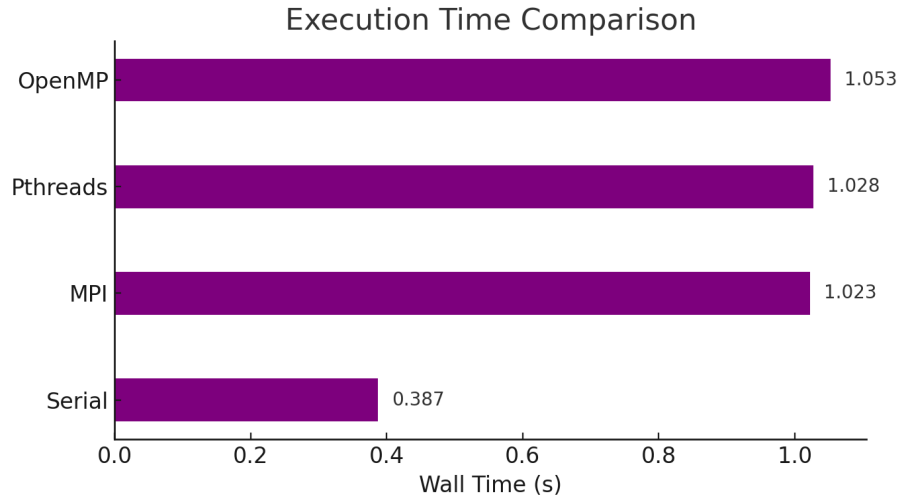
2. 程式設計說明

版本	關鍵實作	工作分配	同步與通訊
Serial	迴圈累積；單執行緒	不分區段	不適用
MPI (4 proc)	`MPI_Comm_rank` 分段；`MPI_Reduce` 合併總和	依 rank 均分 4 份	網路/共享記憶體複製 + Collective
Pthreads (4 thr)	主執行緒 `pthread_create` 四次	依 thread id 均分	`pthread_mutex` + local partial sum
Pthread-Race	與 Pthreads 相同，但 **省略鎖**	同上	無同步 → 競爭條件
OpenMP (4 thr)	`#pragma omp parallel for reduction(+:sum)`	編譯器自動分派	隱含 barrier + reduction

3. 實驗結果與效能分析

方法	近似值	Wall Time (s)	Speed-up*
Serial	3.124451	**0.387**	1.00
MPI	3.124451	1.023	0.38
Pthreads	3.124451	1.028	0.38
OpenMP	3.124451	1.053	0.37
Pthread-Race	1097839 / 4000000	–	–

$*Speed\text{-}up = T_{\text{serial}} / T_{\text{parallel}}$



3.1 效能觀察

1. **序列最快**：在少核心、計算密度偏低情況下，平行管理開銷 (thread spawn/join, barrier, 通訊) 反而拖慢執行。
2. **MPI 無加速**：
   - 單節點多行程需額外複製緩衝區；
   - `MPI_Reduce` 集合通訊成瓶頸。
3. **Pthreads vs. OpenMP**：
   - Pthreads 額外 mutex 鎖，OpenMP 則隱含 barrier；兩者開銷相近。
4. **Amdahl's Law**：
   - 可平行比例 β≈1\beta \approx 1，但管理時間 ToverheadT_{\text{overhead}} 佔比過高，故實測 Speed-up < 1。

4. Race Condition 反思

- `pthread-race.c` 未保護共享變數 `sum`，多執行緒寫入導致 **Lost Update**：結果僅約理論值 2.7 %。
- **解決方案**：
  - 使用 `pthread_mutex_lock` / `pthread_spin_lock`；
  - 或採 **每執行緒私有區段 + 最終匯總**（OpenMP `reduction`、MPI `Reduce`）。
- **以下為Race Condition主要發生於程式碼之位置**：

```
// 每個執行緒要做的事情：重複執行 sum += 1
void* thread_func(void* arg) {
    for (long long i = 0; i < NITERS; i++) {
        // 此處沒有任何同步保護
        // 可能導致多個 thread 同時讀取 / 寫入 sum，產生不預期結果
        sum += 1;
    }
    pthread_exit(NULL);
}
```

---

5. 結論與改進建議

5.1 結論

- **正確性**：除競爭案例外，四版皆符合理論預期。
- **效能**：在本題與硬體設定下，平行化 **未帶來加速**；反映「平行 ≠ 必然加速」。
- **學習重點**：
  1. **同步機制不可或缺**。
  2. **Overhead** 是影響小規模平行程式效能的主因。

5.2 改進方向

面向	建議
計算密度	提高 NN 或引入更複雜函數，使計算時間 ≫\gg 管理開銷。
OpenMP 調度	`schedule(static, large_chunk)` 或 `guided` 以減少 barrier 數。
MPI 架構	節點內先 thread-level reduction，再跨節點 `MPI_Reduce`。
硬體加速	GPU (CUDA) 或 SIMD 指令可大幅提升 FLOPS。
數值精度	改用 `long double` 並增大截斷範圍 L>1000L>1000 減少系統誤差。