# 分布式内存环境下 PSRS 排序算法实验报告

## 一、 实验目的

1. **掌握分布式并行算法设计**：理解并实现并行正则采样排序（PSRS）算法，解决内存不共享环境下的排序问题。
2. **网络编程实践**：利用 Socket 编程接口实现多机间的数据通信与同步。
3. **性能评估与分析**：探究在不同数据规模（1K - 100K）和不同节点数量下的算法性能，分析网络通信开销对加速比的影响。

## 二、 算法设计与实现

### 2.1 算法原理 (PSRS)

PSRS 算法是一种适合分布式存储系统的排序算法，主要分为四个阶段：

1. 局部排序 ： 每个处理器将分配到的数据片进行局部排序。
2. 正则采样 ： 每个处理器从排好序的数据中选取 P−1个样本，发送给主节点。
3. 主元选择与广播 ： Master 收集所有样本，排序后选取 P−1个主元，广播给所有处理器。
4. 全局交换与归并 ： 每个处理器根据主元将数据分段，交换给对应的目标处理器，最后在本地归并排序。

### 2.2 系统架构设计

本实验采用 **Master-Worker** 架构。为了简化全互连通信的复杂性，采用了 **星型拓扑** 进行数据交换：

- **Master (Rank 0)**：负责协调、分发数据、收集样本、计算主元，并在“全局交换”阶段充当路由器，收集所有 Worker 的分桶数据并重新分发。
- **Worker (Rank 1~P-1)**：负责接收数据、计算、发送样本和最终排序。

### 2.3 关键代码解析

#### 2.3.1 通信协议封装

为了保证数据在网络传输中的完整性和端序一致性，封装了 send_vec 和 recv_vec，统一使用大端序。

```
// 封装 Socket 发送 vector，包含长度头和网络字节序转换
static bool send_vec(sock_t s, const vector<int>& a) {
    if (!send_int(s, (int32_t)a.size())) return false;
    vector<int32_t> tmp(a.begin(), a.end());
    for (auto& x : tmp) x = (int32_t)htonl(x); // 主机序转网络序
    return send_all(s, tmp.data(), tmp.size() * sizeof(int32_t));
}
```

### 2.3.2 基于 Master 中转的全局交换

这是代码中最核心的通信部分。Master 收集所有人的桶，然后按目标索引聚合，再发回给对应的节点。

```
// Master 端逻辑
// 1. 接收 Worker 的桶
vector<Buckets> wB(workers);
for (int i = 0; i < workers; ++i) recv_buckets(cli[i], wB[i]);

// 2. 数据重组 (All-to-All 逻辑)
vector<vector<int>> Final(p);
// 处理 Master 自己的数据
for (int k = 0; k < p; ++k) {
    auto bk = extract_bucket_k(myB, k);
    Final[k].insert(Final[k].end(), bk.begin(), bk.end());
}
// 处理 Worker 的数据
for (int i = 0; i < workers; ++i) {
    for (int k = 0; k < p; ++k) {
        auto bk = extract_bucket_k(wB[i], k);
        Final[k].insert(Final[k].end(), bk.begin(), bk.end());
    }
}

// 3. 分发最终数据片
for (int i = 0; i < workers; ++i) send_vec(cli[i], Final[i+1]);
```

### 2.3.3 计时策略

针对小数据量（1K, 5K），通信建立和销毁的波动较大。实验中采用了循环放大策略：

- 对 N=1000，循环跑 500 次取平均值。
- 对 N=100000，跑 1 次。

这保证了计时结果能真实反映算法的通信与计算开销，排除了系统抖动的干扰。

# 三、 实验结果

实验测试了数据规模 N∈{1K,5K,10K,100K}，以及 Worker 数量为 1, 2, 4, 8 时的性能。

编译运行方法：

```
g++ -O3 -std=gnu++17 psrs_socket.cpp -lws2_32 -o psrs_socket.exe
.\psrs_socket.exe master 0.0.0.0 9000 4 100000 3 # 1K、5K、10K、100K
.\psrs_socket.exe worker 127.0.0.1 9000
.\psrs_socket.exe worker 127.0.0.1 9000
.\psrs_socket.exe worker 127.0.0.1 9000
.\psrs_socket.exe worker 127.0.0.1 9000
```

```
PS C:\Users\IScream\Desktop\study\并行分布式计算> .\psrs_socket.exe master 0.0.0.0 9000 8 100000 3 # 1K、5K、10K、100K
[master] listening on 0.0.0.0:9000, waiting 8 workers...
[master] worker #1 connected
[master] worker #2 connected
[master] worker #3 connected
[master] worker #4 connected
[master] worker #5 connected
[master] worker #6 connected
[master] worker #7 connected
[master] worker #8 connected
[master] PSRS run 1/3 (loops=1) ... done. avg time=0.003709000 s
[master] PSRS run 2/3 (loops=1) ... done. avg time=0.003488000 s
[master] PSRS run 3/3 (loops=1) ... done. avg time=0.004000000 s
------------------------------------------------
PSRS Median Tp = 0.003709000 s
[master] Running Serial Benchmark (1 loops/rep)...
Serial Median T1 = 0.005060000 s
Speedup (T1/Tp)  = 1.3642
------------------------------------------------
```

## 3.1 完整测试数据汇总表

| 数据规模 (N) | Worker数 | 总进程数 (P) | 串行耗时 T1 (s) | 并行耗时 Tp (s) | 加速比 |
|---|---|---|---|---|---|
| **1,000** | 1 | 2 | 0.000009 | 0.000246 | **0.038** |
| | 2 | 3 | 0.000007 | 0.000405 | 0.017 |
| | 4 | 5 | 0.000009 | 0.000619 | 0.014 |
| | 8 | 9 | 0.000010 | 0.001099 | 0.010 |
| | | | | | |
| **5,000** | 1 | 2 | 0.000136 | 0.000429 | 0.318 |
| | 2 | 3 | 0.000168 | 0.000480 | **0.350** |
| | 4 | 5 | 0.000165 | 0.000682 | 0.242 |
| | 8 | 9 | 0.000150 | 0.001197 | 0.126 |
| | | | | | |

| 数据规模 (N) | Worker数 | 总进程数 (P) | 串行耗时 T1 (s) | 并行耗时 Tp (s) | 加速比 |
|---|---|---|---|---|---|
| **10,000** | 1 | 2 | 0.000317 | 0.000555 | **0.570** |
| | 2 | 3 | 0.000397 | 0.000706 | 0.562 |
| | 4 | 5 | 0.000370 | 0.000895 | 0.413 |
| | 8 | 9 | 0.000372 | 0.001324 | 0.281 |
| | | | | | |
| **100,000** | 1 | 2 | 0.005002 | 0.008016 | 0.624 |
| | 2 | 3 | 0.008210 | 0.006007 | 1.367 |
| | 4 | 5 | 0.005047 | 0.003225 | **1.565** |
| | 8 | 9 | 0.005060 | 0.003709 | 1.364 |

## 3.2 结果分析

根据结果表，可以观察到以下趋势：

1. **极小规模 (N=1K, 5K) —— 通信开销占主导**
   - 在这些规模下，加速比均远小于 1（即并行比串行慢）。
   - **趋势**：随着 Worker 数量增加（从 1 到 8），Tp耗时反而显著增加（例如 N=1K 时，8 Worker 的耗时是 1 Worker 的 4 倍以上）。
   - **原因**：建立 Socket 连接、三次握手以及多次数据收发的网络延迟都在毫秒级或几百微秒级，而几千个整数的内存排序仅需几十微秒。通信成本完全掩盖了计算收益。
2. **中等规模 (N=10K) —— 盈亏平衡点的逼近**
   - 加速比提升至 0.5 左右，但仍未超过 1。
   - 此时计算量增加，部分抵消了通信的固定开销，但由于数据量仍不够大，网络传输耗时依然是瓶颈。有趣的是，Worker=1 和 Worker=2 的性能接近，说明此时多一个节点带来的计算优势刚好被多一份通信开销抵消。
3. **大规模 (N=100K) —— 并行优势显现与瓶颈**
   - **Worker=1**：加速比 0.62。说明单机通过 Socket 自发自收数据造成的开销约为 40% 的性能损失。
   - **Worker=2**：加速比跃升至 **1.37**。此时总算力增加（3个进程），计算任务被有效分摊，且通信拥堵尚不严重。
   - **Worker=4**：达到本次实验的**最佳性能**，加速比 **1.57**。此时计算负载均衡与通信开销达到了最佳平衡点。
   - **Worker=8**：加速比回落至 **1.36**。

# 四、 实验总结

本次实验成功实现了基于 Socket 的分布式 PSRS 排序算法。

与实验一（OpenMP 多线程）相比，Socket 分布式版本的加速比明显更低。这是因为：

- **多线程**：共享内存，无数据复制，通信开销几乎为零。
- **分布式**：内存不共享，必须序列化数据并通过网络发送，带宽和延迟是主要制约因素。

通过本实验，我对计算通信比这一并行计算核心指标有了直观且深刻的理解。