**计算机体系结构挑战赛设计报告**

题目一：前缀和 (Prefix Sum)

算法设计

采用分块并行前缀和算法，基于Brent-Kung算法实现。该算法将大规模数组分解为多个块，分别在每个块内并行计算前缀和，然后计算块总和的前缀和，最后将块偏移量加到各个块的结果中。

内核函数设计

// 块内前缀和计算

\_\_global\_\_ void blockPrefixSum(int\* input, int\* output, int\* blockSums, int N)

// 块总和前缀和计算

\_\_global\_\_ void blockSumPrefix(int\* blockSums, int\* blockOffsets, int numBlocks)

// 偏移量加法

\_\_global\_\_ void addBlockOffsets(int\* output, int\* blockOffsets, int N)

参数配置

块大小：1024线程/块

网格大小：根据输入大小动态调整

共享内存：每块4KB共享内存

性能优化

内存访问优化：合并内存访问，减少bank冲突

计算并行化：充分利用GPU的并行计算能力

边界处理：正确处理数组边界和不完整块

题目二：Softmax函数

算法设计

采用数值稳定的Softmax算法，通过最大值平移避免数值溢出问题。

算法流程

最大值查找：找到输入向量中的最大值

指数计算：计算每个元素的指数值（减去最大值）

求和计算：计算指数和

归一化：将每个指数值除以总和

内核函数设计

// 最大值查找

\_\_global\_\_ void findMaxKernel(const float\* input, float\* maxValue, int N)

// 指数和计算

\_\_global\_\_ void computeExpSumKernel(const float\* input, float\* expSum, float maxVal, int N)

// Softmax计算

\_\_global\_\_ void softmaxKernel(const float\* input, float\* output, float maxVal, float sum, int N)

题目三：全对最短路径 (APSP)

算法设计

采用分块Floyd-Warshall算法，将距离矩阵分块处理，利用GPU的并行计算能力加速传统Floyd-Warshall算法。

算法流程

初始化阶段：构建初始距离矩阵

分块处理：将矩阵分为32×32的块

三层迭代：更新枢轴块，更新行块和列块，更新剩余块

结果输出：输出最终距离矩阵

内核函数设计

// 枢轴块更新

\_\_global\_\_ void updatePivotBlock(int V, int B, int k, int\* dist)

// 行块和列块更新

\_\_global\_\_ void updateRowColBlocks(int V, int B, int k, int\* dist, bool isRow)

// 剩余块更新

\_\_global\_\_ void updateOtherBlocks(int V, int B, int k, int\* dist)

总结

本次挑战赛完成了三个GPU加速算法的实现方案，每个方案都针对具体问题的特点进行了优化设计，充分利用了GPU的并行计算能力，同时保证了算法的正确性和数值稳定性。