Contents

1 PRAM **6**
- 1.1 Prerequisites . . . 6
- 1.2 Definition . . . 6
- 1.3 How it works . . . 7
  - 1.3.1 Computation . . . 7
  - 1.3.2 PRAM Classificiation . . . 7
  - 1.3.3 Strengths of PRAM . . . 8
  - 1.3.4 How to compare PRAM models . . . 8
- 1.4 MVM algorithm . . . 10
- 1.5 SPMD sum . . . 12
- 1.6 MM algorithm . . . 16
- 1.7 PRAM variants and Lemmas . . . 17
- 1.8 PRAM implementation . . . 18
- 1.9 Amdahl's and Gustafson's Laws . . . 20

2 Fundamentals of architecture **23**
- 2.1 Introduction . . . 23
  - 2.1.1 Simplest processor . . . 23
  - 2.1.2 Superscalar processor . . . 24
  - 2.1.3 Single Instruction, Multiple Data (SIMD) processor . . . 25
  - 2.1.4 Multi-Core Processor . . . 25
- 2.2 Accessing Memory . . . 26
  - 2.2.1 What is a memory? . . . 26
  - 2.2.2 How to reduce processor stalls . . . 28
    - 2.2.2.1 Cache . . . 28
    - 2.2.2.2 Multi-threading . . . 28

3 Programming models **31**
- 3.1 Implicit SPMD Program Compiler (ISPC) . . . 31
- 3.2 Shared Address Space Model . . . 35
- 3.3 Message Passing model of communication . . . 36
- 3.4 Data-Parallel model . . . 37

4 Parallel Programming Models and pthreads **39**
- 4.1 How to create parallel algorithms and programs . . . 39
- 4.2 Analyze parallel algorithms . . . 41
- 4.3 Technologies . . . 44
- 4.4 Threads . . . 47
  - 4.4.1 Flynn's taxonomy . . . 47
  - 4.4.2 Definition . . . 47
  - 4.4.3 pthreads API . . . 49
    - 4.4.3.1 Creation . . . 49
    - 4.4.3.2 Termination . . . 50
    - 4.4.3.3 Joining . . . 51
    - 4.4.3.4 Detaching . . . 52
    - 4.4.3.5 Joining through Barriers . . . 53
    - 4.4.3.6 Mutexes . . . 54
    - 4.4.3.7 Condition variables . . . 54

5 OpenMP v5.2 **55**
5.1 Introduction 55
5.2 Basic syntax 57
5.3 Work sharing 60
5.3.1 For 60
5.3.1.1 Reduction 65
5.3.2 Sections 67
5.3.3 Single/Master 68
5.3.4 Tasks 69
5.3.4.1 Task dependences 72
5.4 Synchronization 76
5.5 Data environment 79
5.6 Memory model 87
5.7 Nested Parallelism 90
5.8 Cancellation 94
5.9 SIMD Vectorization 97

6 GPU Architecture **100**
6.1 Introduction 100
6.2 GPU compute mode 101
6.3 CUDA 103
6.3.1 Basics of CUDA 103
6.3.2 Memory model 107
6.3.3 NVIDIA V100 Streaming Multiprocessor (SM) 109
6.3.4 Running a CUDA program on a GPU 112
6.3.5 Implementation of CUDA abstractions 118
6.3.6 Advanced thread scheduling 121
6.3.7 Memory and Data Locality in Depth 126
6.3.8 Tiling Technique 135
6.3.8.1 Tiled Matrix Multiplication 138
6.3.8.2 Implementation Tiled Matrix Multiplication . . 143
6.3.8.3 Any size matrix handling 148
6.3.9 Optimizing Memory Coalescing 153

7 CUDA **162**
7.1 Introduction 162
7.2 CUDA Basics 166
7.2.1 GPGPU Best Practices 168
7.2.2 Compilation 170
7.2.3 Debugging 172
7.2.4 CUDA Kernel 175
7.3 Execution Model 178
7.4 Querying Device Properties 180
7.5 Thread hierarchy 182
7.6 Memory hierarchy 185
7.7 Streams 193
7.8 CUDA and OpenMP or MPI 197
7.8.1 Motivations 197
7.8.2 CUDA API for Multi-GPUs 202
7.8.3 Memory Management with Multiple GPUs 205

7.8.4 Batch Processing and Cooperative Patterns with OpenMP 211
7.8.5 OpenMP for heterogeneous architectures 213
7.8.6 MPI-CUDA applications 216

8 Memory Consistency 220
8.1 Coherence vs Consistency 220
8.2 Definition 223
8.3 Sequential Consistency Model 225
8.4 Memory Models with Relaxed Ordering 229
8.4.1 Allowing Reads to Move Ahead of Writes 230
8.4.2 Allowing writes to be reordered 232
8.4.3 Allowing all reorderings 234
8.5 Languages Need Memory Models Too 236
8.6 Implementing Locks 238
8.6.1 Introduction 238
8.6.2 Test-and-Set based lock 240
8.6.3 Test-and-Test-and-Set lock 244

9 Heterogeneous Processing 248
9.1 Energy Constrained Computing 250
9.2 Compute Specialization 251
9.3 Challenges of heterogeneous designs 265
9.4 Reducing energy consumption 268

10 Patterns 271
10.1 Dependencies 271
10.2 Parallel Patterns 280
10.2.1 Nesting Pattern 281
10.2.2 Serial Control Patterns 282
10.2.3 Parallel Control Patterns 284
10.2.4 Serial Data Management Patterns 289
10.2.5 Parallel Data Management Patterns 292
10.2.6 Other Parallel Patterns 295
10.3 Map Pattern 297
10.3.1 What is a Map? 297
10.3.2 Optimizations 299
10.3.2.1 Sequences of Maps 299
10.3.2.2 Code Fusion 300
10.3.2.3 Cache Fusion 301
10.3.3 Related Patterns 302
10.3.4 Scaled Vector Addition (SAXPY) 304
10.4 Collectives operations 307
10.4.1 Reduce (or Reduction) Pattern 308
10.4.2 Scan Pattern 314
10.5 Gather Pattern 325
10.5.1 What is a Gather? 325
10.5.2 Shift 329
10.5.3 Zip 331

Index 330

10.5.3 Zip

The **Zip operation** is a special case of the gather pattern where **two (or more) arrays are combined by interleaving their elements**. It functions like a zipper, **taking one element from each array in sequence to form a new combined array**. It is important to note that it **works with different types**, so we can zip elements of different types, like integers and floats, or even complex objects.

How does it work?

The operation **takes an element from the first array**, then **one from the second** array, another from the third, and so on, and **repeats the process**. The **output is the combined sequence**.

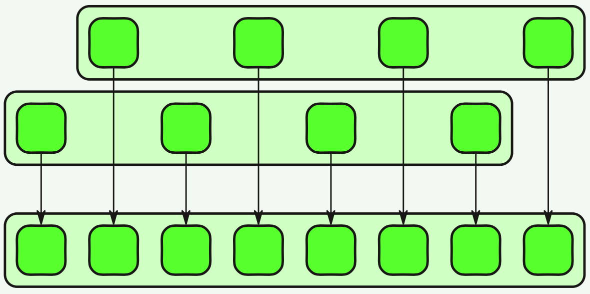
Example 46: zip operation

Consider two arrays:

1. Real Parts: contains real numbers.
2. Imaginary Parts: contains imaginary numbers.

The output is a combined sequence like:

```
[Real0, Imag0, Real1, Imag1, Real2, Imag2, ...]
```



Parallelism

Each pair of elements (one from each array) can be **combined independently**. This independence allows **parallel execution since there's no dependency between the operations for different pairs**.