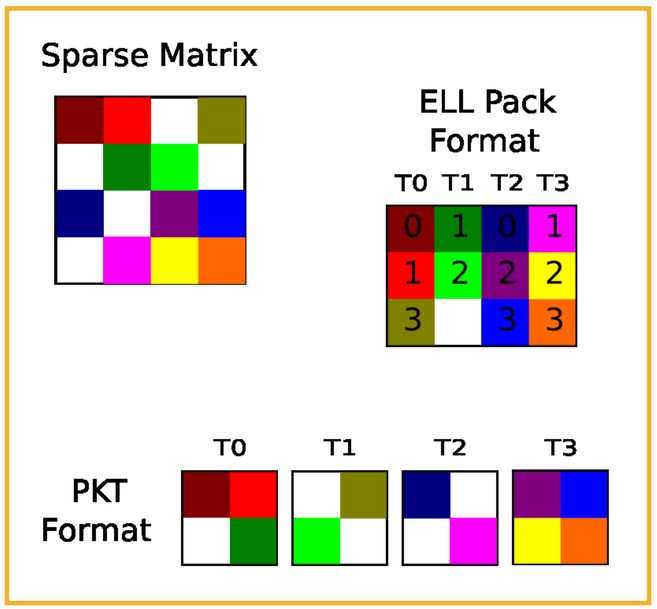
Sparse Matrix Vector Multiplication on GPUs

Ozan Demirlioglu

MOTIVATION

Sparse matrix vector multiplication is an important kernel in scientific computing. As there are necessarily very few flops per memory access, it makes sense to implement it on the GPU, which has excellent memory bandwidth, despite its need for very specific access patterns.

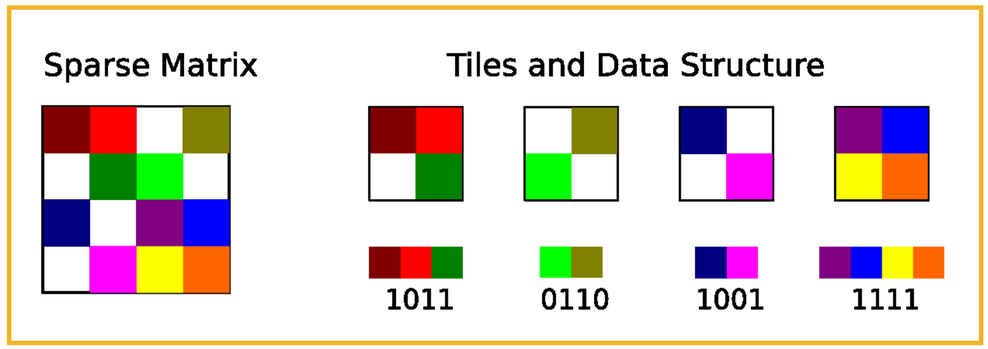


PREVIOUS WORK

- CUDPP (from NVIDIA)
  - Segmented Scan Implementation
  - Recursive Calls move data multiple times
- IBM
  - Matrix Based Segmented Scan (Faster)
  - Tiled Implementation
    - Fixed Size Tiles
    - Tiles Are Aligned
    - Some Zero Fill
- NVIDIA
  - ELL pack format
    - Like CSR, data interleaved for better memory access
    - Irregular access to source vector
    - Texture caching
  - PKT format
    - Similar to IBM Tile Implementation
    - Dense storage for tiles

DESIGN

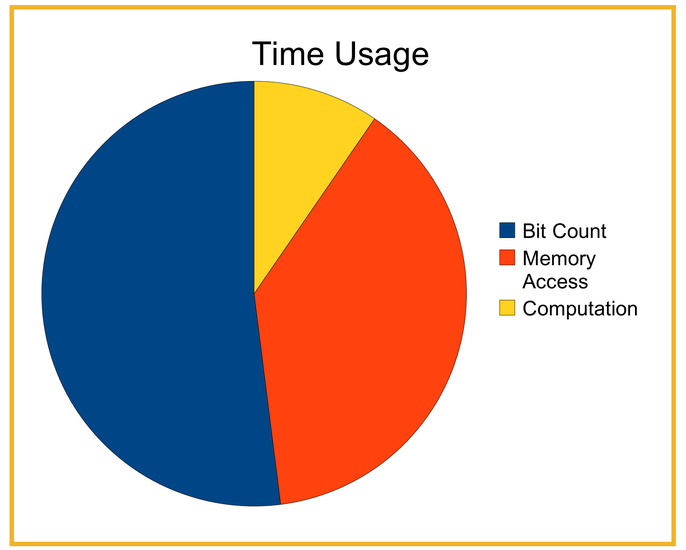
- Data Structure
  - Sparsely stored tiles, using bitmasks
    - Tiles allow re-use of source vector
    - Sparse storage means less zero fill
    - Bitmasks used to expand tiles in registers
    - Bitmask is tiny amount of extra storage
  - Tiles are fixed size and aligned
    - Fixed size avoids load imbalance in computation blocks
    - Alignment ensures coalesced accesses to global memory
    - Each tile creates small amount of overhead



- Algorithm
  - GPU processor reads in a single block's metadata, then source vector
  - Data is read into a cache in blocks
    - Columns are smaller than the blocks
    - Use of sliding window to control cache
    - Each thread processes a single row within the computation block
  - Synchronization
    - No synchronization required within each computation block
    - Several methods for synchronization across blocks
      - Atomic updates to destination vector
        - Atomic writes are slow, even without conflicts
      - Bind a GPU processor to a row of tiles
        - Easy to create load imbalance
      - Process blocks in multiple stages
        - Time delay between stages may make it impractical

RESULTS

- Sources of Slowdown
  - Bit counting takes about half the time
    - No native bit count operation
  - Small tile sizes make source/destination vector accesses expensive (1/9th of data)
- Bandwidth
  - Achieving about 80% of bandwidth when bit counting is removed
  - Currently, 88% of bandwidth is the source matrix
    - Remainder is reading source vector and reading/writing destination vector
  - No uncoalesced memory accesses



FUTURE WORK

- Improve bit counting methods
  - Perhaps precomputing bitcounts is better than doing it on the fly
- Reduce zero fill
  - METIS tried, just made things worse
  - BFS to move data points closer together
- Benchmark different forms of synchronization
- Autotuning
  - Current tile size is 32x32, but can be adjusted to any multiple thereof
  - Number of threads per GPU can be adjusted