GPUalg projects WS2024

Efficient encoding and application of a scan on binary data

Task

- Input: M[] binary flags
- Output: P[] index vector used to permute data
- Computation: P= scan(M)

Typical application: packing of data

- Input: src[] some input data
- Output: dst[k] packed data
- Computation: dst[k] contains the k-th value src[i] for which M[i] is true

Performance problem

- P[] contains int32_t or int64_t indices and is thus 32 or 64 times bigger than M[]
- We need a much more efficient encoding of the scan result as a mapping $F(i)=k$ or $F^{-1}(k)=i$

Solution

- Don't store P[] but rather a hierarchy of coarser vectors
- For simplicity we will use at most two levels `vector<H> vh; vector<G> vg;`
- Implement a `class ScanBool<nhl,ngl,H,G>` with member functions `setup(M); apply(src, dst);` , with defaults `H=uint16_t, G=Index`
- `nh=(1<<nhl)` says how much coarser vh[] is in comparison to M[]
- `ng=(1<<ngl)` says how much coarser vg[] is in comparison to vh[]

1) Forward mapping $F(i)=k$

Existing setup

- Input: M[] binary flags
- Output: Psrc[] index vector used to permute data
- Computation: Psrc= exclusive_scan(M)

Existing application

- Input: src[] some input data
- Output: dst[] packed data
- Computation: scatter `if(M[i]) dst[Psrc[i]]= src[i]`

New setup

- Input: M[] binary flags
- Output: Psrc[] index vector used to permute data
- Computation: `ScanBool<nhl,ngl,H,G> sb; sb.setup(M);`

New apply

- Input: src[] some input data
- Output: dst[] packed data
- Computation: `ScanBool<nhl,ngl,H,G> sb; sb.apply(src, dst);`

Milestones

1. Thrust implementation of setup and apply
2. Benchmarking and plot generation
3. New setup and apply with two levels
4. New setup and apply with one level
5. Comparison of thrust and new solution

2) Backward mapping $F^{-1}(k)=i$

Existing setup

- Input: M[] binary flags
- Output: Pdst[] index vector used to permute data
- Computation: Pdst= copy_if(identity, M)

Existing application

- Input: src[] some input data
- Output: dst[] packed data
- Computation: gather `dst[k]= src[Pdst[k]]`

New setup

- Input: M[] binary flags

- Output: Psrc[] index vector used to permute data
- Computation: `ScanBool<nhl,ngl,H,G> sb; sb.setup(M);`

New apply

- Input: src[] some input data
- Output: dst[] packed data
- Computation: `ScanBool<nhl,ngl,H,G> sb; sb.apply(src, dst);`

Milestones

1. Thrust implementation of setup and apply
2. Benchmarking and plot generation
3. New setup with two levels
4. New apply with two levels
5. Comparison of thrust and new solution