Using Futhark for a fast, parallel implementation of forward and back projection in algebraic reconstruction methods - A pre-study

Lærke Pedersen and Mette Bjerg Lindhøj

University of Copenhagen

08/11/2018

SIRT

Solve the problem:

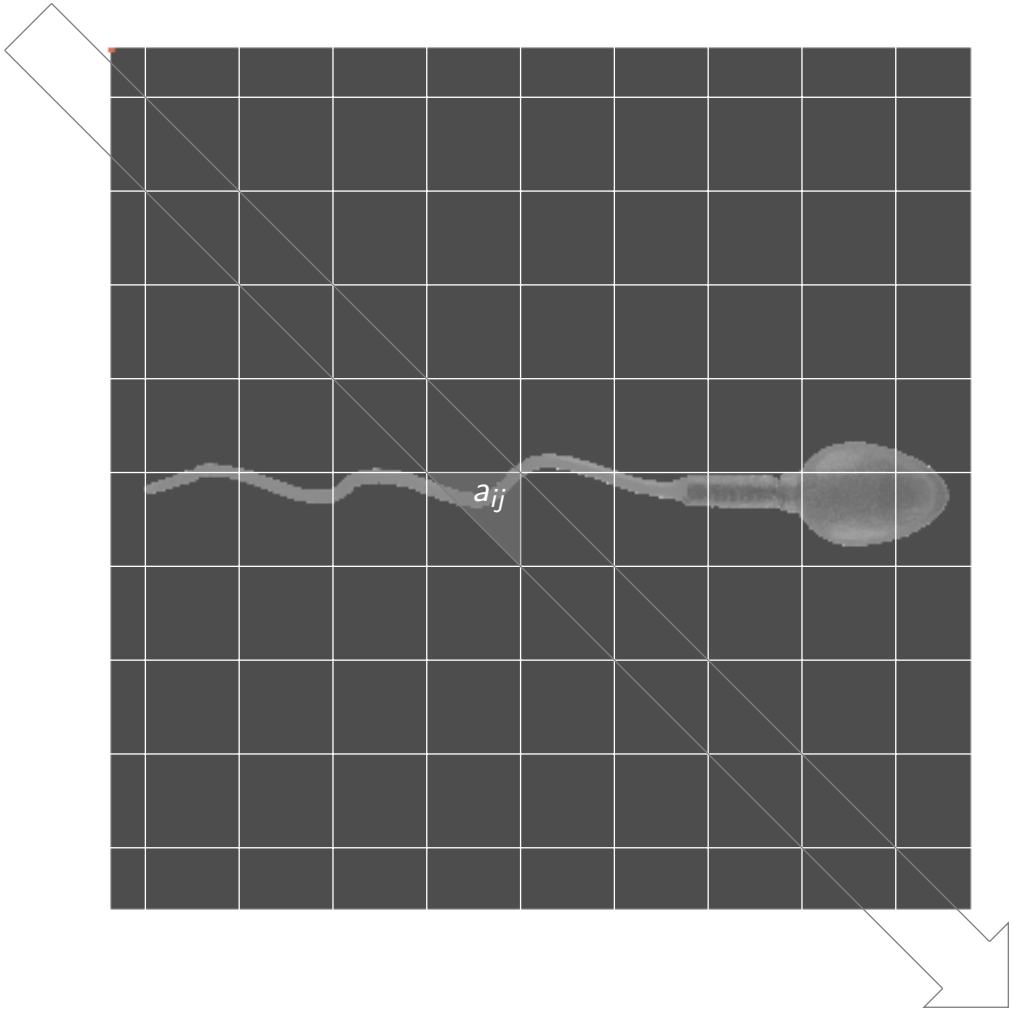
$$\boldsymbol{f}^* = argmin_{\boldsymbol{f}} \|\boldsymbol{p} - \boldsymbol{A}\boldsymbol{f}\| \tag{1}$$

iteratively using this update step:

$$\boldsymbol{f}^n = \boldsymbol{f}^{(n-1)} + \boldsymbol{C}\boldsymbol{A}^T\boldsymbol{R}(\boldsymbol{p} - \boldsymbol{A}\boldsymbol{f}^{(n-1)}), \tag{2}$$

where $\boldsymbol{C}$ and $\boldsymbol{R}$ are the diagonal matrices containing the inverse column and row sums of the system matrix respectively.

The system matrix



The problem is in the size

- Consider reconstructing a single slize of a volume from a detector of size $n \times n$
- The number of rays is n
- The number of angles is $\frac{n \cdot \pi}{2}$
- A typical value for n is 2048
- In semi sparse format the matrix will take up $2 \cdot 4 \cdot 2048 \cdot \lceil \frac{2048 \cdot \pi}{2} \rceil \cdot (2 \cdot 2048 - 1) \approx 216 GB$

System matrix computation

```
1 for ray = 0; ray < numberofrays; ray++ //parallel
2   for pixel = 0; pixel < pixels; pixel++ //parallel
3     if ray intersects pixel:
4         (p1,p2) = intersectionpoints pixel ray
5         A[ray][pixel] = distance p1 p2
```

Figure: $W(r, n) = O(r \cdot n^2)$, $D(r, n) = O(1)$

```
1 for ray in rays //parallel
2   while (isingrid focuspoint) //seq
3         pixel = calculatepixel focuspoint ray
4         nextpoint = findnextpoint focuspoint ray
5         A[ray][pixel] = distance nextpoint focuspoint
6         focuspoint = nextpoint
```

Figure: $W(r, n) = O(r \cdot n)$, $D(r, n) = O(n)$

Flattening

Utilizing nested parallelism is notoriously difficult, flattening the code can give marked improvements. However, flattening is also a way to systematically reason about nested code, as well as transforming it. Our project is currently a mix of flattened and nested code.

I will look into flattening our code in an example of one of our primary work areas, forward projection

The format of the matrix

```
1 [[(0.01,0),(0.04,3),(0.01,2),(-1.0,-1)],
2 [ (0.10,0),(0.40,3),(0.05,2),( 1.0,-1)],
3 [ (0.01,1),(0.04,5),(-1.,-1),(-1.0,-1)],
4 [...                                   ],
5 [...                                   ],
6 [...                                   ]]
```

Figure: An example of the matrix output format.

Flattening

Forward projection is handled as a matrix vector multiplication. We utilize both a nested version and a flattened version.

```
1    map (\row -> reduce (+) 0 <| map (\(v, ind) -> unsafe
         (if ind == -1 then 0.0 else v*vect[ind])) row )
       mat_vals
```

This translates to, still nested;

```
1    map(\row ->
2      let (ts, fs) = partition2 (\(_, ind) -> != -1) row
           Work(n), depth(log(n)) --if-then-else
3      let multiplied = map (\(v, ind) -> v*vect[ind]) ts
           work(n)
4      in reduce (+) 0 multiplied Work(n), depth(log(n))
5    ) mat_vals
```

Flattening

```
1     map (\row -> reduce (+) 0 <| map (\(v, ind) -> unsafe
          (if ind == -1 then 0.0 else v*vect[ind])) row )
        mat_vals
```

- A flattened matrix enforces coalesed reads from memory
- In SIMD a if-then-else is executed by evaluating the if, executing the then brach while the rest of the cores wait, then the same for the else branch. Partition means coalesed reads going forward
- The cons of flattening are that it may require impractically high memory-usage which is already an issue in our problem, and does not account for locality of reference.

Code transformations

- By dumping the compiled code with the –dump command and getting the time spend in different kernels with the -D option we found that most of the time was spend on calculating the system matrix
- Futhark can not merge a map with a loop
- The code had a lot of branching
- We tried to mitigate this by removing as much from the loop as possible, and reorganising branches.

Changing the algorithm

```
1  for ray in rays //parallel
2    for i=-halfsize; i < halfsize; i++ //parallel
3          (l1,pixel1) = intersection1 ray i
4          (l2,pixel2) = intersection2 ray i
5          A[ray][pixel1] = l1
6          A[ray][pixel2] = l2
```

Figure: $W(r, n) = O(r \cdot n)$, $D(r, n) = O(1)$

Flattening the new algorithm

```
let weights_doublepar(angles: []f32) (rays: []f32) (gridsize: i32): [][](f32,i32) =
    let halfgridsize = gridsize/2
    let entryexitpoints =  convert2entryexit angles rays (r32(halfgridsize))
    in map(\(ent,ext) -> (flatten(map (\i ->
                calculate_weight ent ext i gridsize
        )((-halfgridsize)...(halfgridsize-1))))) entryexitpoints
```

```
let weights_doublepar_flat [n] [m](angles: [n]f32) (rays: [m]f32) (gridsize: i32): [](
    f32,i32) =
    let halfgridsize = gridsize/2
    let entryexitpoints =  convert2entryexit angles rays (r32(halfgridsize))
    let replicatedentries = mapreplicate gridsize entryexitpoints
    let replicatedcolumns = flatten(replicate (n*m) ((-halfgridsize)...(halfgridsize
        -1)))
    let flatgridEntryexit = zip replicatedentries replicatedcolumns
    in flatten(map(\((p1,p2),i) ->
                calculate_weight p1 p2 i gridsize
        ) flatgridEntryexit)
```

Figure: Flattening the new algorithm.


Hao Gao.
Fast parallel algorithms for the x-ray transform and its adjoint.
Medical physics, 39(23127102):7110–7120, November 2012.


Y. Long, J. A. Fessler, and J. M. Balter.
3d forward and back-projection for x-ray ct using separable footprints.
IEEE Transactions on Medical Imaging, 29(11):1839–1850, Nov 2010.


F. Natterer.
The Mathematics of Computerized Tomography.
Society for Industrial and Applied Mathematics, 2001.


Z. Xue, L. Zhang, and J. Pan.
A new algorithm for calculating the radiological path in ct image reconstruction.
In *Proceedings of 2011 International Conference on Electronic Mechanical Engineering and Information Technology*, volume 9, pages 4527–4530, Aug 2011.