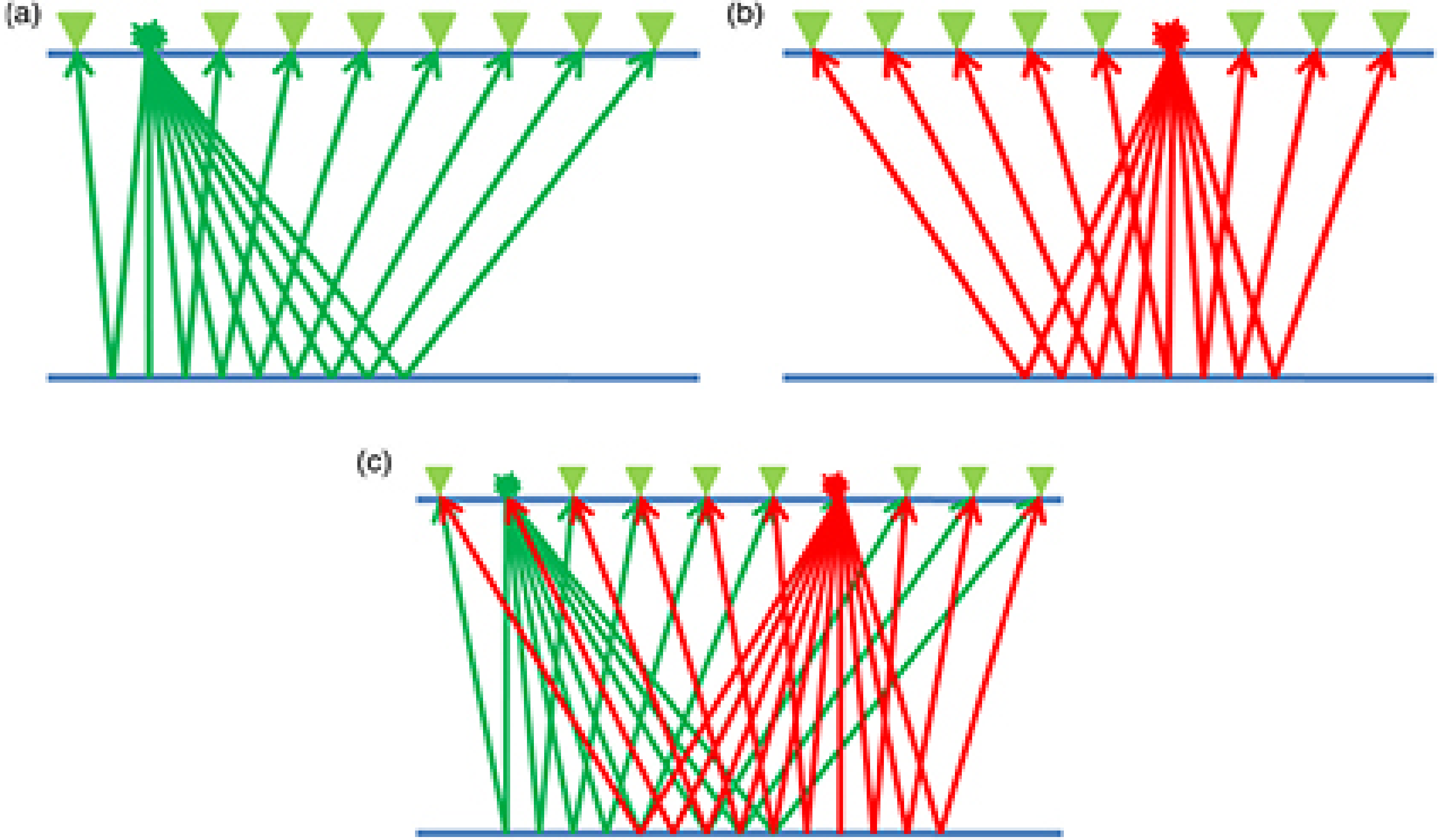
HPC methods for high resolution Radon transform and deblending

Kai Zhuang, Daniel Trad

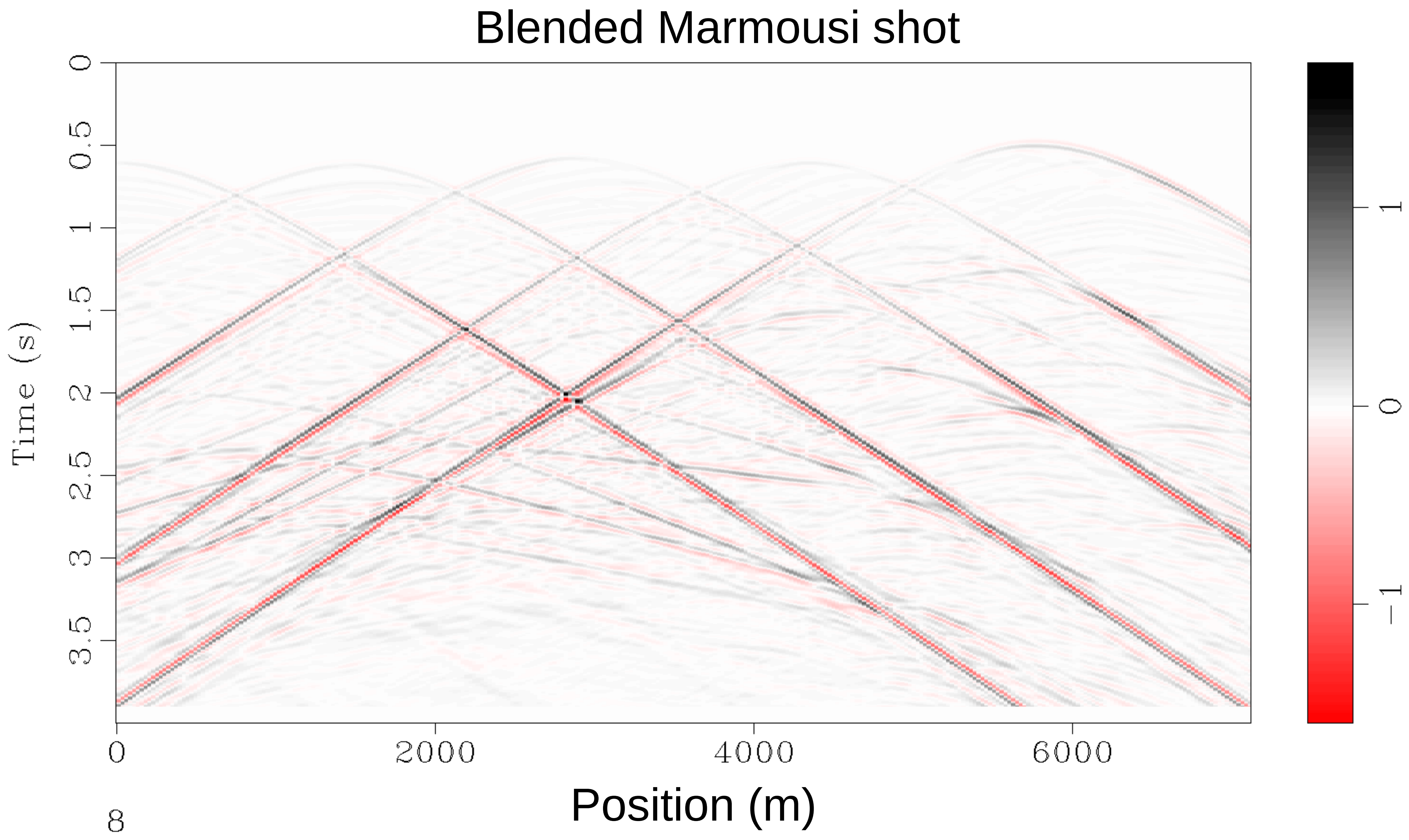
CREWES Annual Meeting

Dec 3 2020

Blending

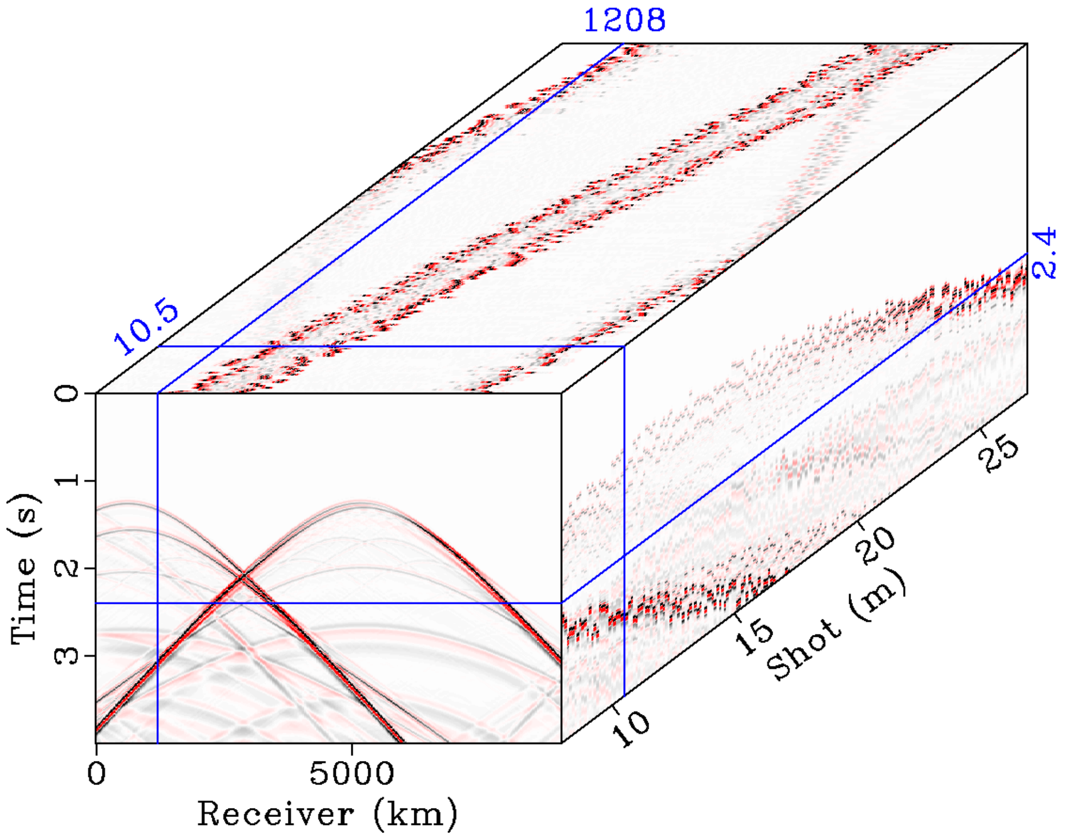


Blended acquisition

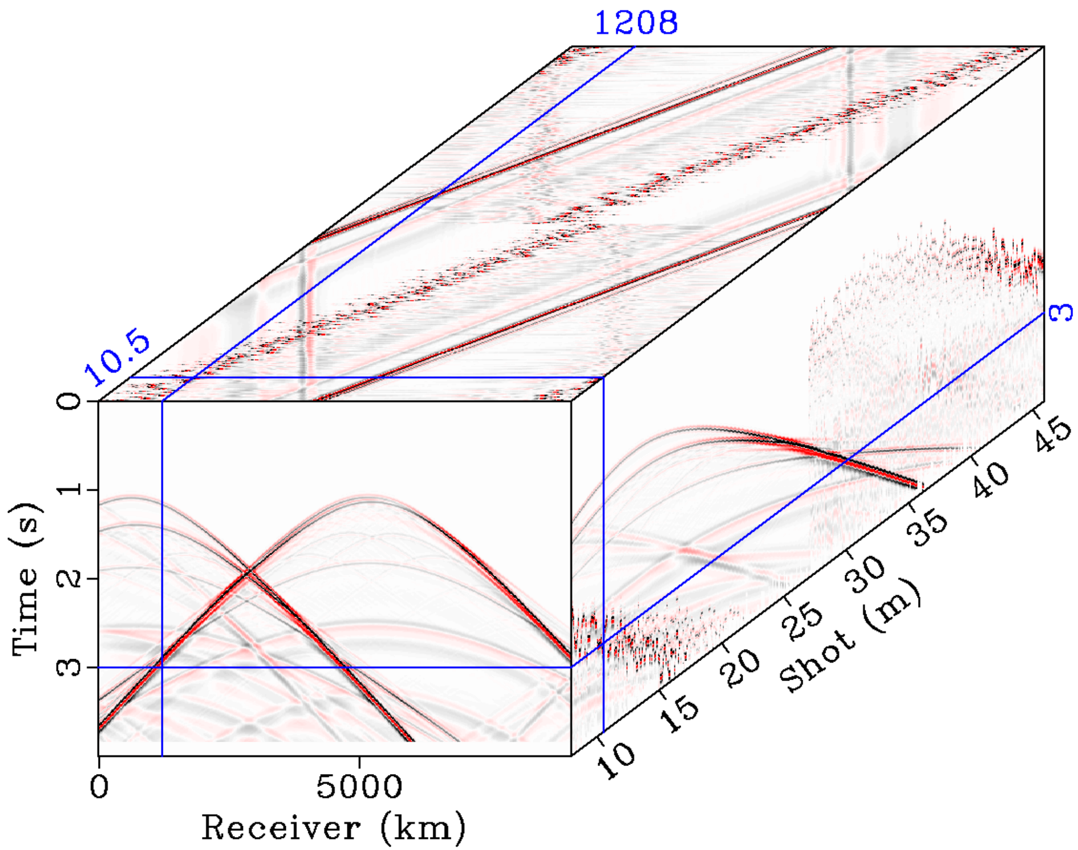


Blending and the importance of time dithering

Blended Data



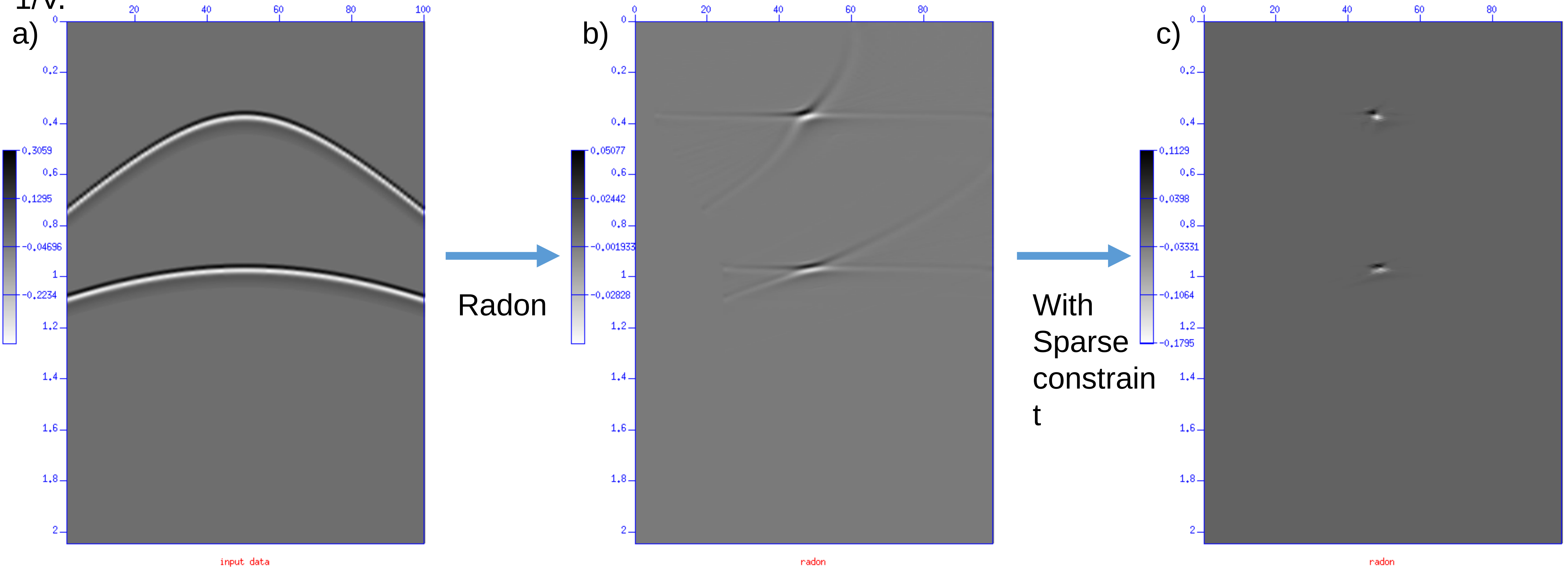
Pseudo Deblended Data



Sparse Hyperbolic Radon Transform

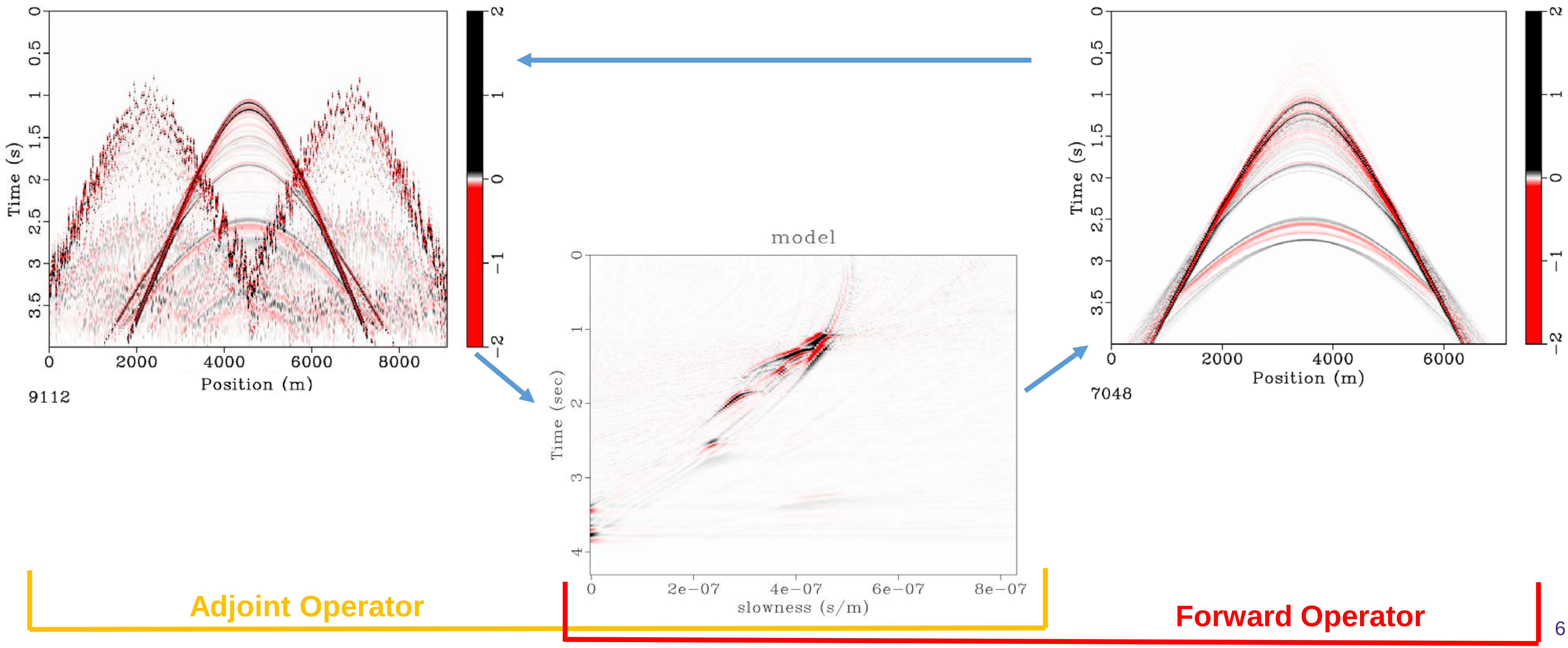
$$u(p,\tau) = \int_{h_1}^{h_2} d(h, t = \sqrt{\tau^2 + p^2 h^2})dh$$

where u(p,t) is the radon space data, p is the slowness, t is the two way travel time, h1 is the upper offset limit, h2 the lower offset limit, and d is the data space to be transformed. The slowness p is then defined as the inverse of velocity 1/V.



Denoising – sparse radon transform

$$\left|\left|S_{pdb} - Rm\right|\right|_2^2 + \mu\left|\left|m\right|\right|_1^1$$



Radon Denoising

$$S_{pdb} = S_{bl}\Gamma^H$$

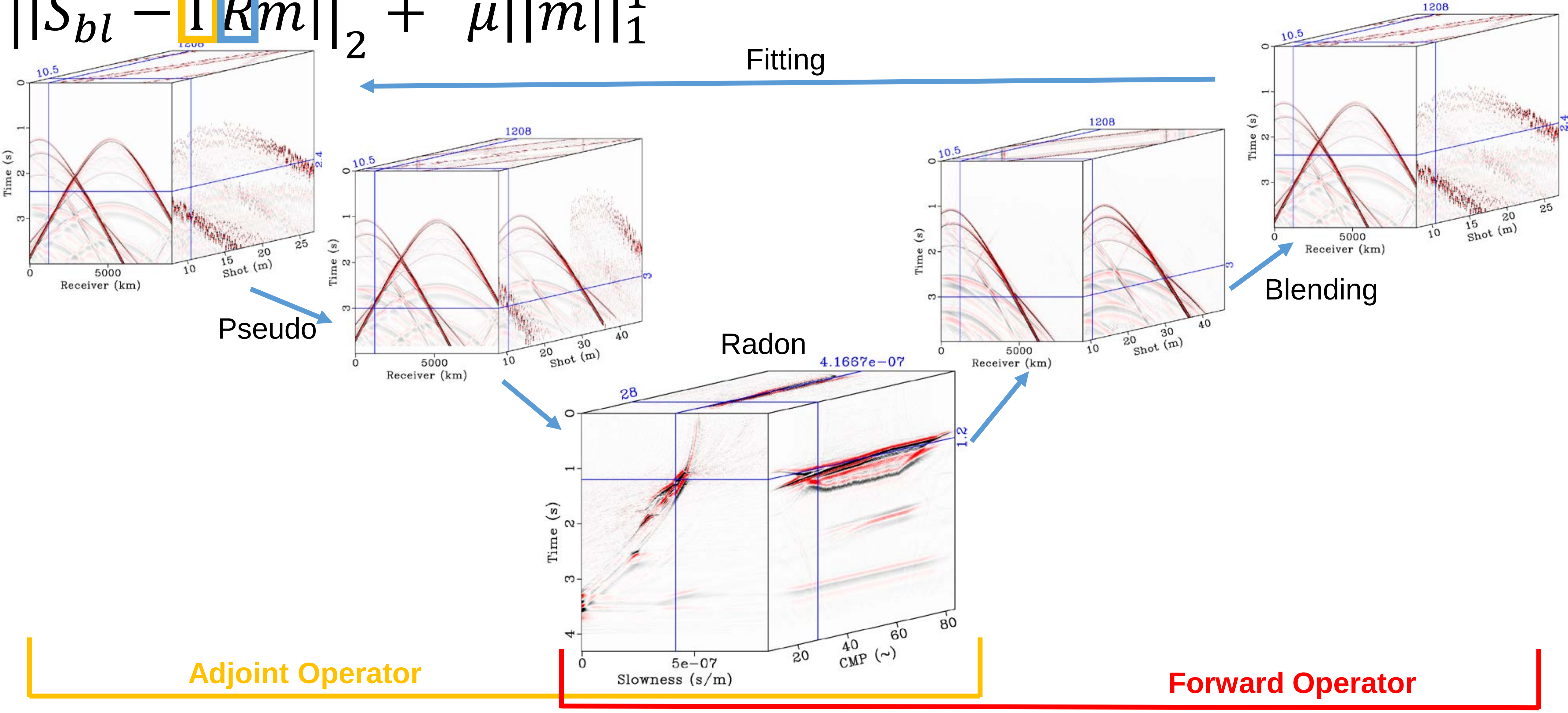
$$||S_{pdb} - Rm||_2^2 + \mu||m||_1^1$$

Radon Inversion

$$||S_{bl} - \Gamma Rm||_2^2 + \mu||m||_1^1$$

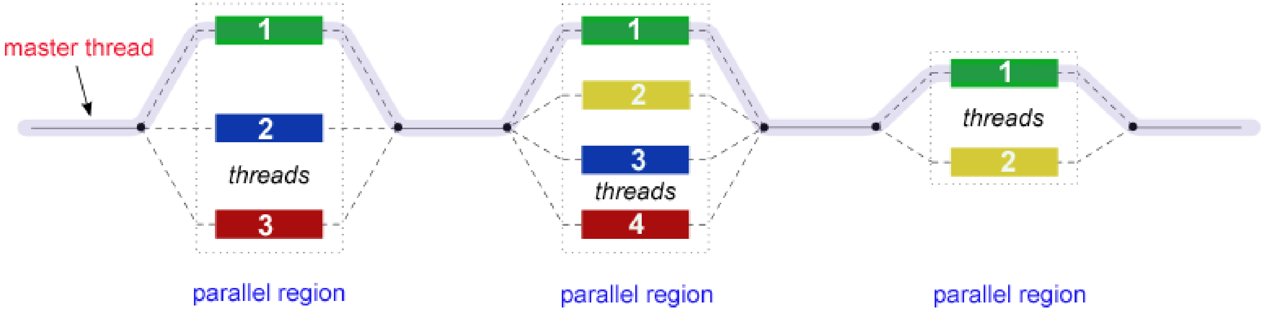
Sparse Inversion

$$||S_{bl} - \Gamma R m||_2^2 + \mu ||m||_1^1$$



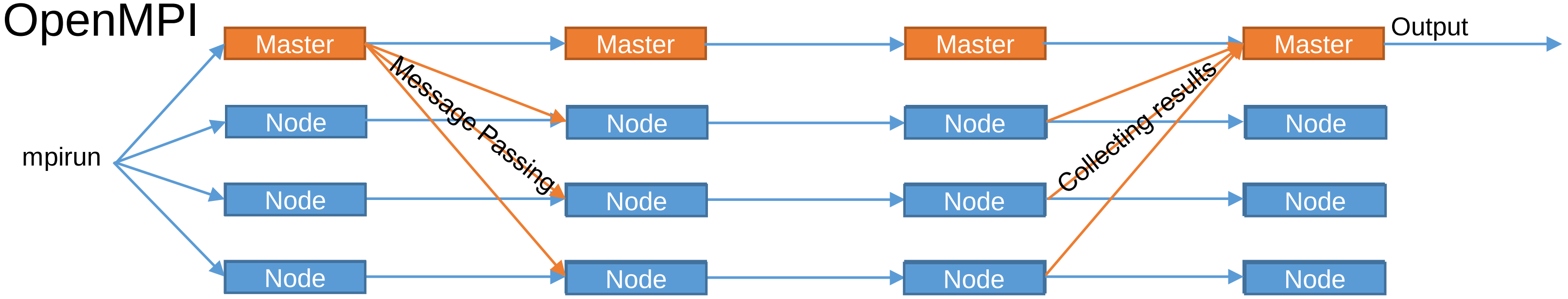
Parallel programming methods

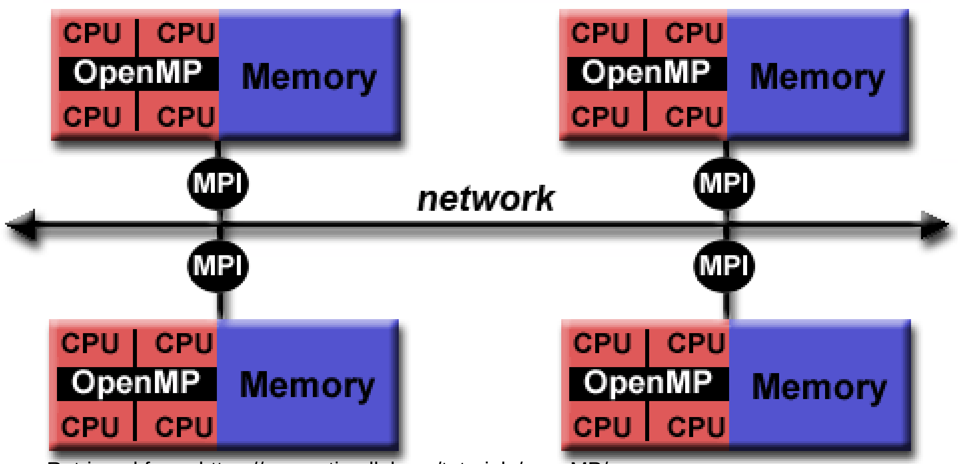
OpenMP



Retrieved from: https://computing.llnl.gov/tutorials/openMP/

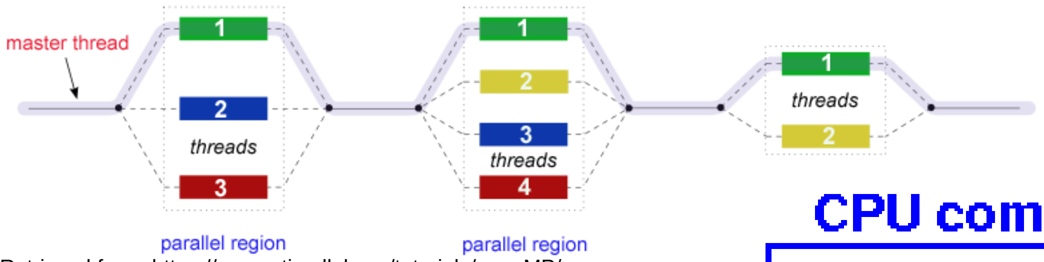
OpenMPI



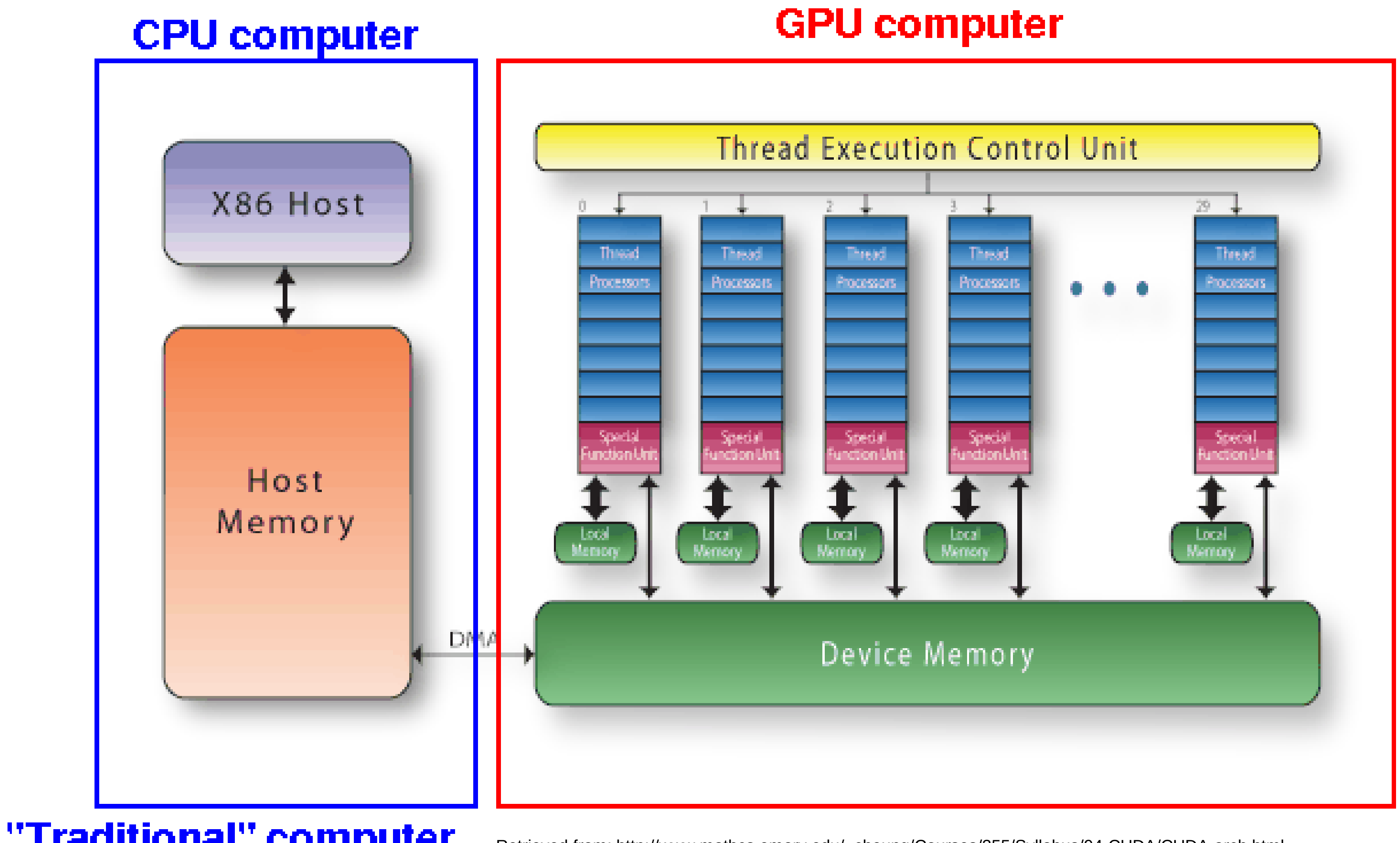


Retrieved from: https://computing.llnl.gov/tutorials/openMP/

CUDA



Retrieved from: https://computing.llnl.gov/tutorials/openMP/



Retrieved from: http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/94-CUDA/CUDA-arch.html

API specifics

Key things to note:

OpenMP

- Simplest to implement limited speedup due to overhead in single threaded portions
- Works on practically any multi-thread CPU

OpenMPI

- Most complex to implement requiring extensive modification to pre-existing code
- Designed originally for multi-server applications
- Can be used on a single system rather than a cluster for significant performance boost in many scenarios
- Main limitation is excessive data transfers slows execution significantly

CUDA

- Very easy to implement but requires a Nvidia GPU
- User must be aware of architecture nuances and limitations
- Most effective when memory access is sequential

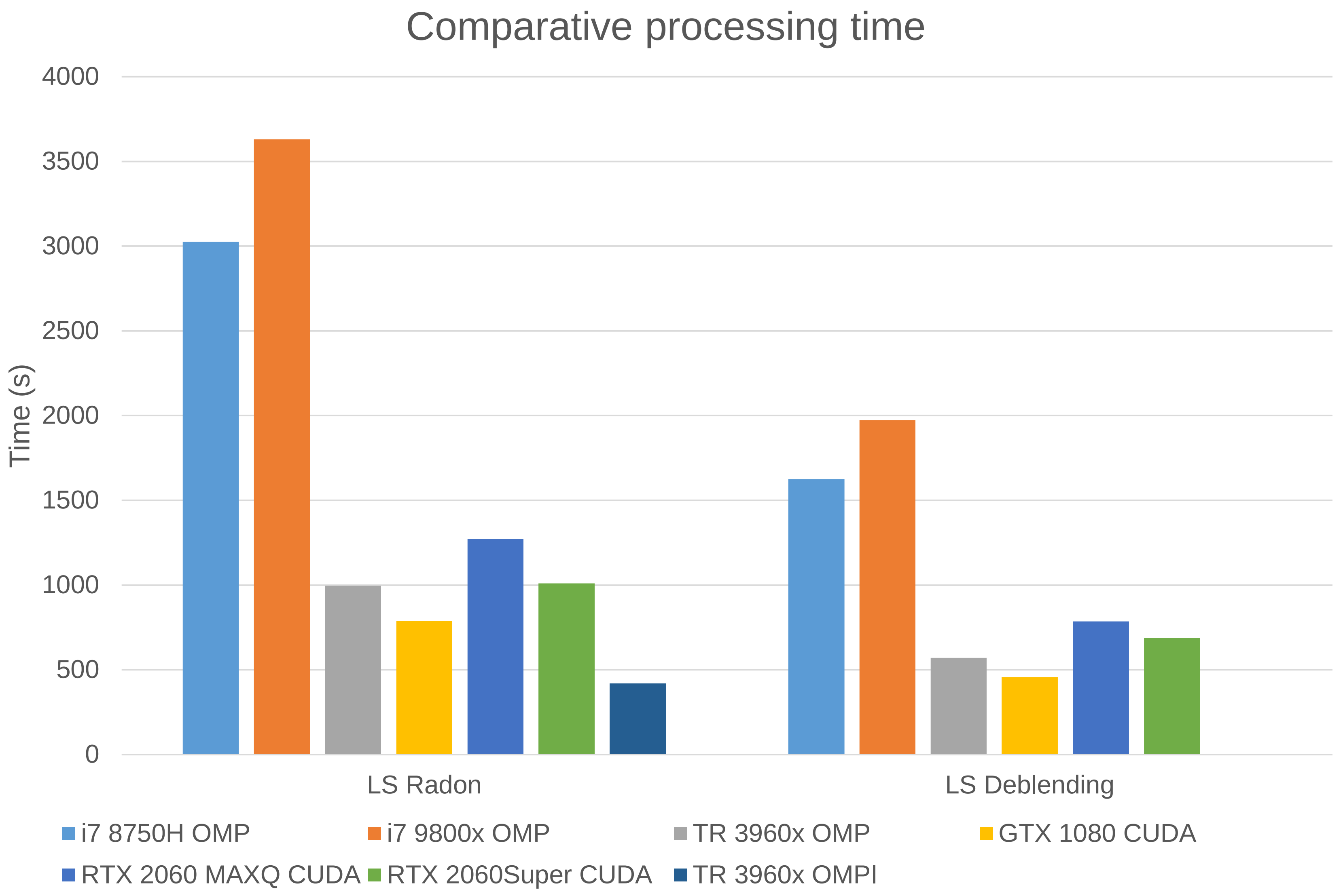
Algorithm 1 Radon pseudo code

function RADON
    #PRAGMA OMP PARALLEL FOR ▷ insert for parallelization of loop
    for q = slowness **do**
        for h = offset **do**
            $moveout = h^2 * q$
            for it = 0 to nt **do**
                $time = \sqrt{(it * dt)^2 + moveout}$
                $model[iqNt + it] += data[ihNt + INT(time/dt)]$
            end for
        end for
    end for
end function

Algorithm 2 CUDA Radon pseudo code

__global__
function CUDA_ RADON
    int it = blockIdx.x * blockDim.x + threadIdx.x
    int iq = blockIdx.y * blockDim.y + threadIdx.y
    int ih = blockIdx.z * blockDim.z + threadIdx.z
    if (iq >= nq || ih >= nh || it >= nt) return
    int ihNt = ih*nt
    int iqNt = iq*nt
    double timemax = dt*nt
    double $moveout = h^2 * q$
    double $time = \sqrt{(it * dt)^2 + moveout}$
    $atomicAdd(\&model[iqNt + it], data[ihNt + INT(time/dt)])$
end function

Performance results



Conclusion

- All APIs have good points and draw backs
- CUDA is easy to implement with significant computational advantages for scientific processing
- GPUs are easier to upgrade and price to performance is better than CPU
- Multi GPU is much easier than multi CPU
- openMP is still easiest to implement and is recommended for small workloads
- openMPI is useable for single system parallelization but uses significant amounts of RAM due to allocation per node

We thank the sponsors of CREWES for continued support. This work was funded by CREWES industrial sponsors, NSERC (Natural Science and Engineering Research Council of Canada) through the grants CRDPJ 461179-13 and CRDPJ 543578-19.