Parallelization

Notes:

- 2D PSF seems to introduce artifacts that are apparent on the coronal view of the reconstructed images that we've deemed as unacceptable.
- 3D PSF correction is therefore required over 2D if the user requires resolution recovery.
  - Without parallelization this approach is too slow for commercial applications.
  - Target reconstruction time is **strictly less than 10 minutes** to be considered performant. **Ideally it would be ~2-5 minutes**.
- Jowett Chan has now prototyped two approaches to achieve CPU-based parallelization, one that he devised and second which was recommended by STIR's Kris Thielemans.
  - We need to determine if these two approaches are technically equivalent to each other, i.e. if they produce the same result, and compare their performance to decide which approach to implement/adopt.
- Regardless of the approach used, we'll need to record the performance across the set of datasets that we currently have access to (~6-8), some with and some without AC maps for attenuation correction and/or masking.
- We'll need to compare the quality of the outputs to that of commercial SPECT systems' reconstruction software. Ideally from several vendors, e.g. **GE, Siemens**, Mediso, Philips, etc.

Methods:

1. <Jowett's approach> Each thread handles a different viewgram independently
   a. The OpenMP version in Visual Studio is older than 201107, which does not support loop collapsing. Therefore, the nested double loop in `distributable.cxx` was combined into a single loop to enable parallelization.
   b. We removed the omp critical in line 898 of [STIR/src/recon_buildblock/ProjMatrixByBinSPECTUB.cxx]() to allow simultaneous execution of wm_calculation by different threads
   c. Each thread declares two new local variables, `local_wm` and `local_wmh`, at the beginning of `ProjMatrixByBinSPECTUB::compute_one_subset`. Then, we replace all the global `this->wm` and `this->wmh` with `local_wm` and `local_wmh` in the `ProjMatrixByBinSPECTUB::compute_one_subset`.
2. <Kris' approach> Parallelize the calculation of weight matrix
   a. Create `local_wm` within `wm_calculation.`
   b. The five nested loops are parallelized using `#pragma omp for schedule(dynamic)`.

c. Merge each thread's local_wm back to the global wm.
d. Disabling the parallelization of the viewgram if we are computing the projection matrix. If not, we allow parallelization of the viewgram which allows parallelization of the step for iterative reconstruction.

Performance Testing:

Compare the two parallelization methods

- Select a single dataset to be used for testing, e.g. TST-837-01 (SPECT/CT Bone Scan)
- For each method, track the execution time and memory consumption under the same recon parameters for the following number of threads:
    - 1, 2, 4, 8, <max> (according to JC's machine)

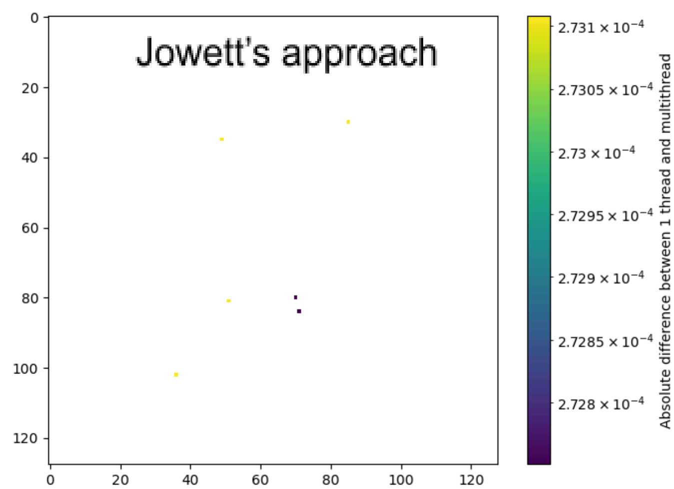
	Parallelize viewgrams			parallelize wm_caclulation		
# of threads	Time [second]	Speed up	Memory [Gb]	Time [second]	Speed up	Memory [Gb]
1	596	1.00	43.221	596	1.00	43.221
2	297	2.00	43.396	354	1.68	43.514
4	156	3.82	43.731	259	2.30	43.832
8	108	5.51	44.038	422	1.41	44.198
16	154	3.87		1400	0.42	

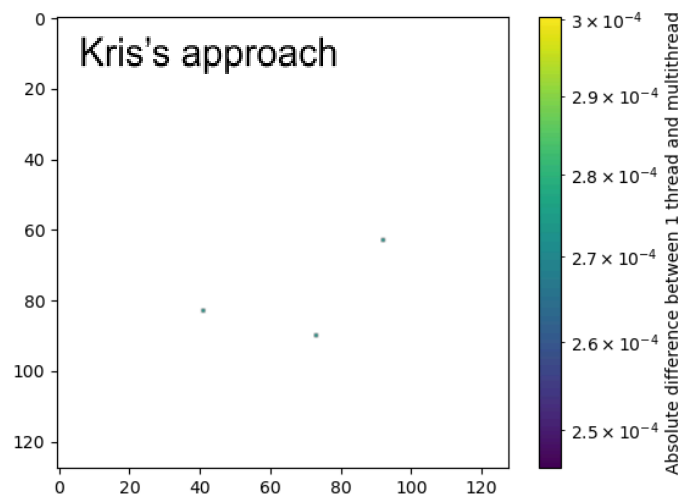
- The above table is obtained by running OSMAPOSL based on the following input
    - Study="TST-837-01"
    - Mask="Attenuation Map"
    - Sigma=1.8
    - PSF="3D"

      Jowett's approach can achieve a maximum speed up of ~5.5x using 8 threads, whereas Kris's approach can achieve a maximum speed up of ~2.3x using 4 threads. The smaller speedup in Kris's approach is due to the overhead of allocating an additional `local_wm` for each thread inside `wm_calculation` during every call. In contrast, Jowett's approach replaces the use of the global `wm` with a `local_wm` for each thread in `compute_one_subset`, avoiding the need to repeatedly allocate additional memory.

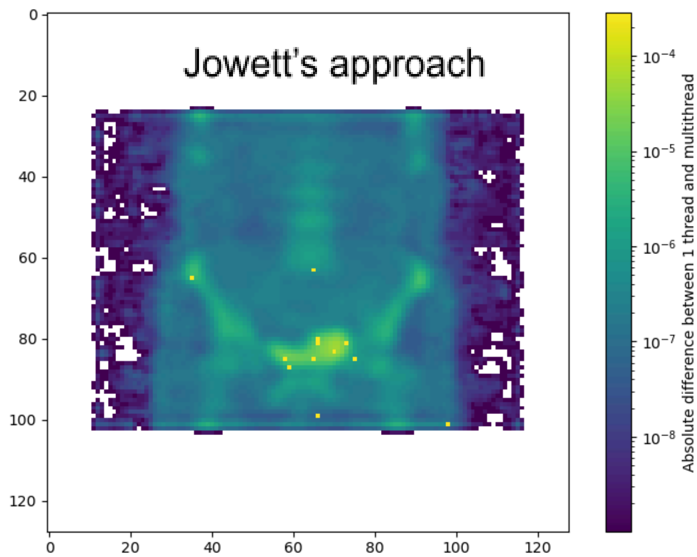
- Plot the results to gauge the difference in performance wrt to memory and compute time.
- Examine difference images for each method compared to a baseline image reconstructed using only a single thread. We expect to see uniform difference images, allowing for low intensity noise.

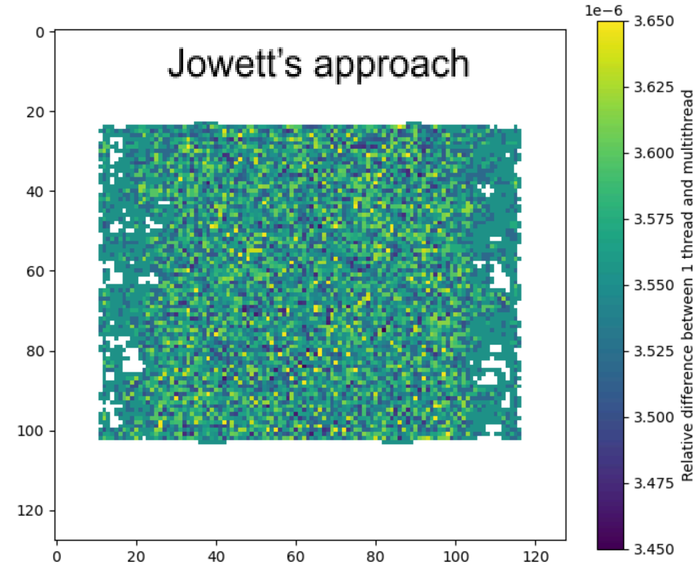
  This image shows the absolute difference, defined as abs(result_multithread - result_singlethread), for one slice [:, 64, :] of the resulting reconstructed image based on the above study and setting. In this test study, we observed randomly placed 2.73e-4 differences between single thread and multithread. Around 99 pixels have these differences within the 128 x 128 x 128 reconstructed image.

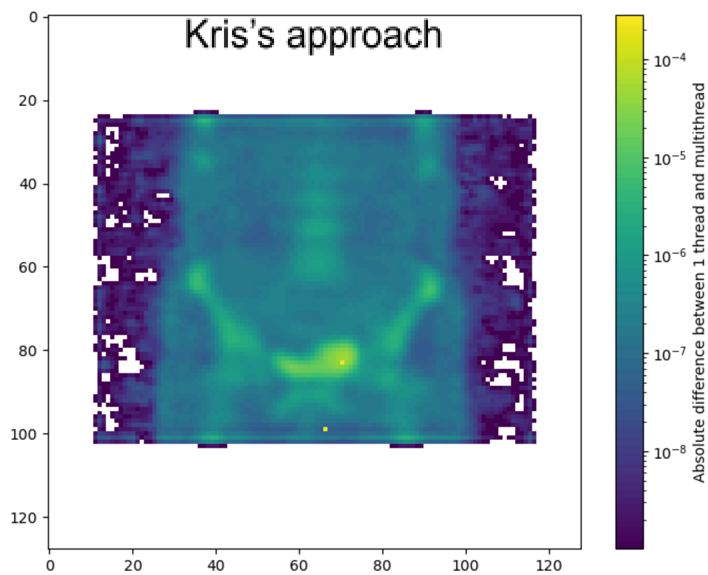


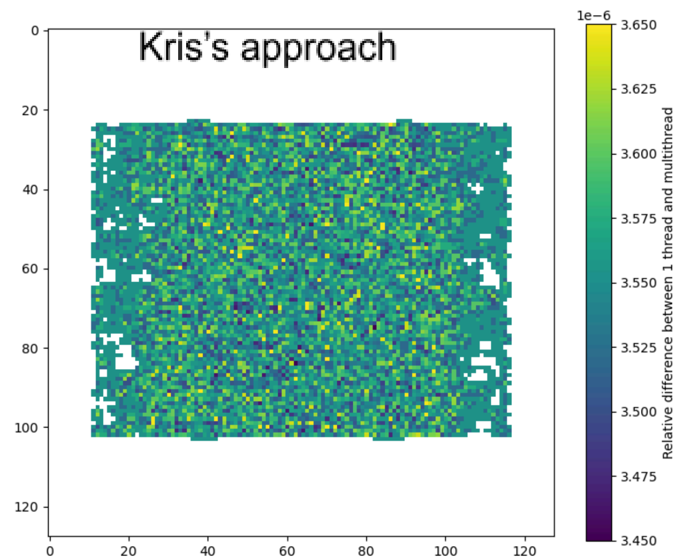


However, if we increased the Sigma from 1.8 to 2.0, the memory consumption increased and we observed absolute differences that scale with the intensity of the reconstructed image. Below shows 4 images, which are the absolute and relative difference based on Jowett's approach in the first row, and that based on Kris's approach in the second row. The absolute difference scales clearly with the intensity of the reconstructed image, but if the relative difference, defined as abs(result_multithread - result_singlethread)/result_singlethread, is small and shows a noise-like pattern. Both absolute and relative differences together suggest that the difference originates from the floating-point rounding error.





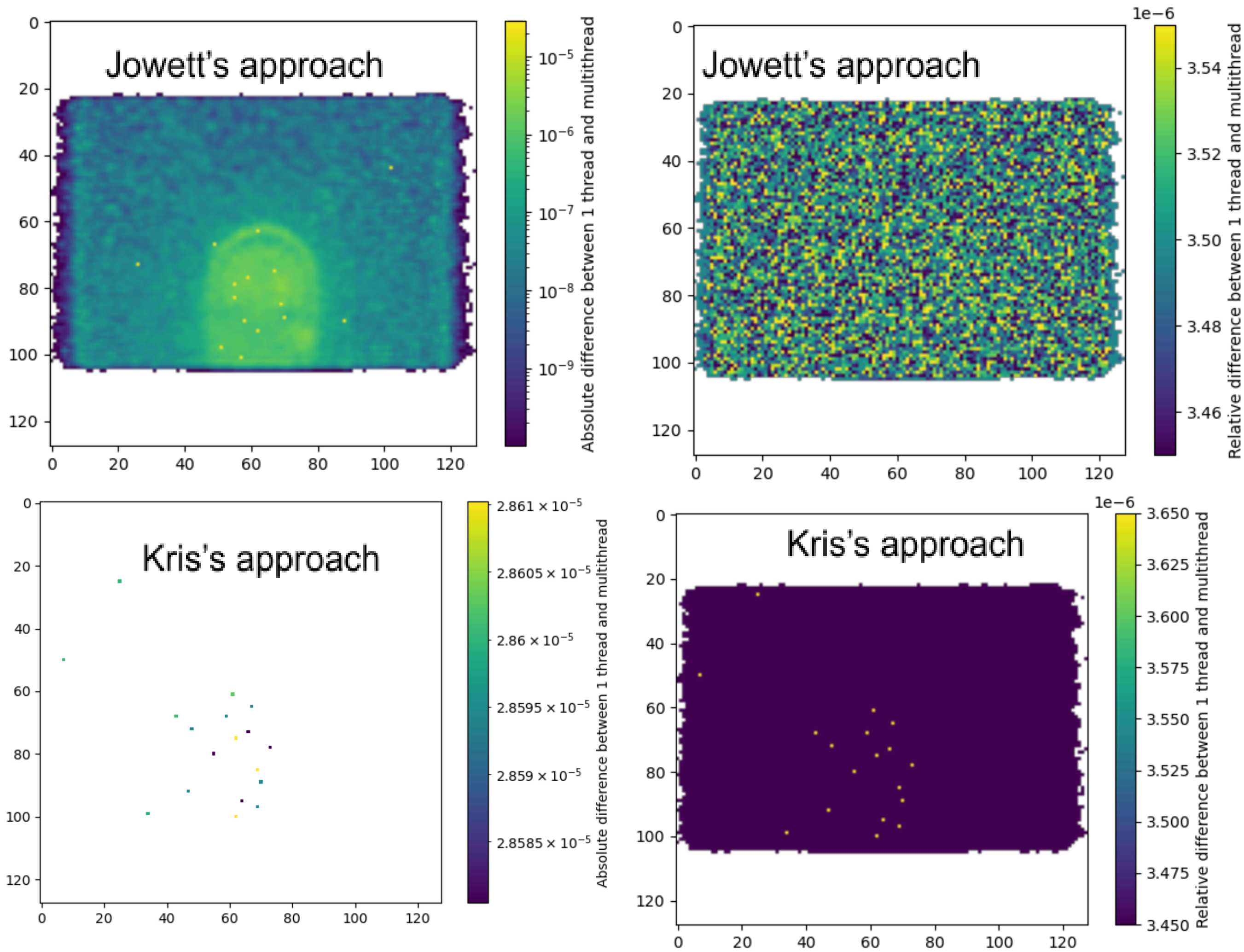




Generate benchmark performance data across a range of SPECT scans using the best performing method from above.

- Record the compute time and memory consumption for SPECT recon with 3D PSF resolution recovery for the following datasets: TST-837-02, 03, 04, 05, etc… could serve as an assessment as to the general performance characteristics across different scan types, scanners and collimators.
    - Study="TST-837-06", Mask="Cylinder", Sigma=1.2, PSF="3D", No AC map

	Jowett's approach			Kris' approach		
# of threads	Time [second]	Speed up	Memory [Gb]	Time [second]	Speed up	Memory [Gb]
1	298	1.00	34.25	298	1.00	34.25
2	148	2.01	34.30	186	1.60	34.45
4	77	3.87	34.51	133	2.24	34.79
8	52	5.73	35.22	135	2.20	35.79
16	52	3.73	35.41	261	1.14	



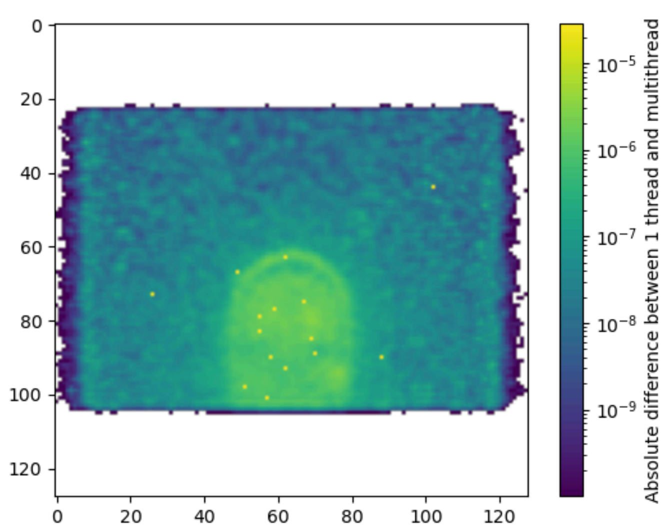
Kri's approach sometimes does not show the kind of difference that scales with the intensity of the image in certain configurations, which means their order of summation follows similarly to that of single thread runs in certain configurations (see section Remaining work).

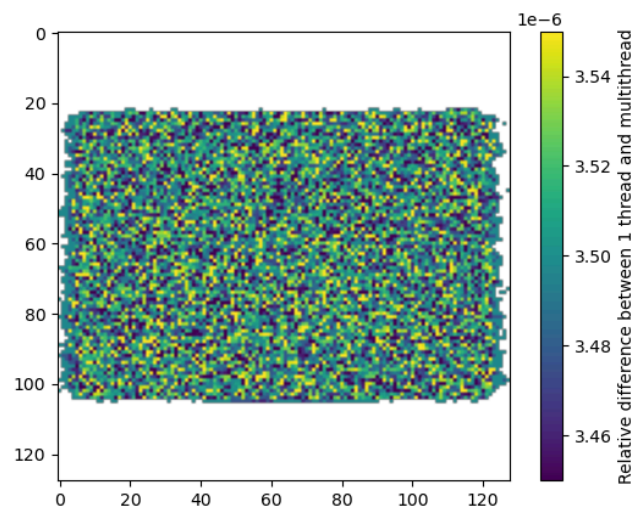
Remaining work:

<Compile a list of issues/artifacts that remain as part of the parallelization task>

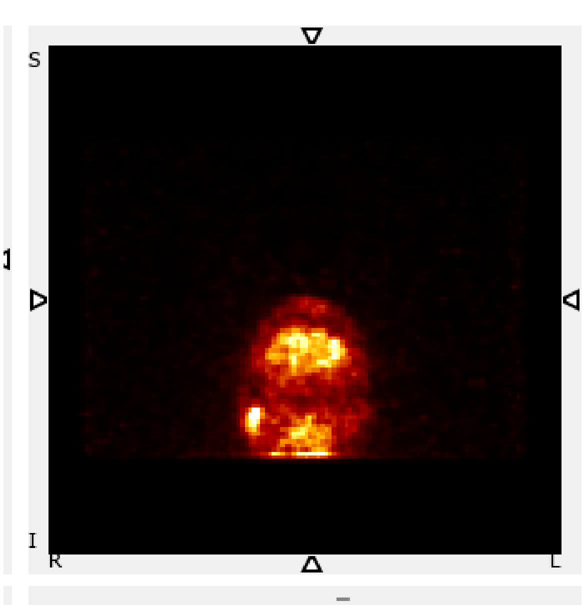
1) Proof of floating-point rounding error
This is to investigate the non-uniformly distributed difference, that also scales with the intensity of the reconstructed image. If the difference is truly a floating point rounding error, we expect to reproduce the difference by re-ordering the summation operation in the code, while keeping the operation logically equivalent. We manage to reproduce the same difference by changing one line of the original code. We remove line 442 in the distributable.cxx
for (int i = 0; i < static_cast<int>(vs_nums_to_process.size()); ++i)
and replace it with
for (int i = static_cast<int>(vs_nums_to_process.size())-1; i>= 0; i--).

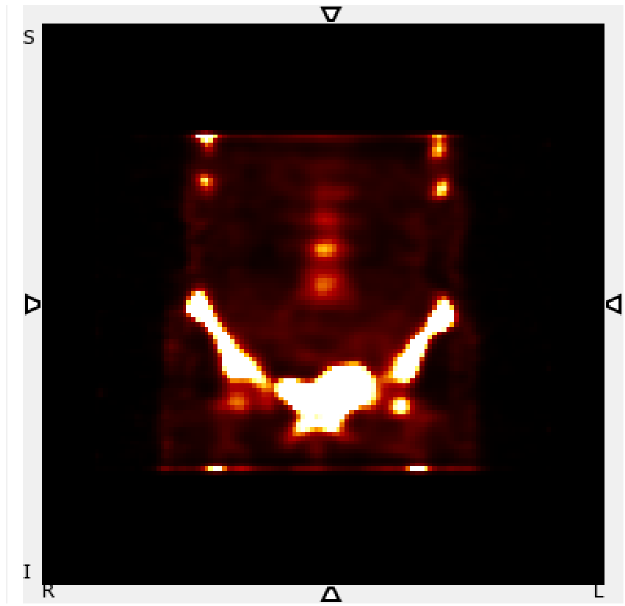
The idea is to flip the loop which iterates from vs_nums_to_process.size() to 0, instead of going from 0 to vs_nums_to_process.size(). Since the order of summation is now reversed, we should expect the floating point error to occur, but their logic are equivalent. We test the code using TST-837-01, Sigma=1.2, 3D psf , a Cylinder mask and , most importantly, using only a single thread. Indeed, the absolute difference shows again a dependence with the intensity of the image, while the relative difference is small and distributed like noise. We, therefore, confidently confirm that the observed difference in the multithreaded runs are as well floating point rounding error.





2) There is an artifact using the 3D PSF model on the border of the coronal and sagittal view, which is not related to multithreading.
    - Its a boundary condition problem. The end of field of view, the reconstruction is usually wrong overthere.
    - Try No mask?





3) check more tests with higher itereations, Ex: using 160 subiterations. Higher iterations should produce better image, but also check it multitherad fails at higher iterations. use different num of subset and check perofrmace. ex:1,2,4,8,12 to check speedup

    To be continue…

4) check convergence with the cost/objective function?
    To be continue…