Eurographics Symposium on Parallel Graphics and Visualization (2020)
S. Frey, J. Huang, F. Sadlo (Editors)

Effective Parallelization Strategies for Scalable, High-Performance Iterative Reconstruction: Supplemental Materials

Christiaan Gribble[1]

[1]Applied Technology Operation, SURVICE Engineering

These supplemental materials provide information for accessing the source code distribution of our prototype XCT reconstruction system, summarize key elements of the code, and present scaling results on the two test platforms omitted from the main text.

Source Code

As noted in Section 3.1 of the main text, we provide the full source code for the prototype XCT system used to explore the impact of thread count and reconstruction volume resolution on performance.

Access to the most recent stable release of the system is available via the project homepage at:

http://www.rtvtk.org/~cgribble/research/pct-egpgv20/

Additionally, read-only access to the development repository is available via HTTP with `git`:

```
git clone http://www.rtvtk.org/code/pct-egpgv20.git
```

Unless otherwise stated directly in the source, this code is distributed under the BSD 3-Clause License. Please see the `LICENSE` file distributed with the source for more information.

The key elements of this source distribution include:

- **`common/`** contains code used in both the reconstruction engine and the supporting applications, including common data structures and mathematics primitives.
- **`engine/`** contains code implementing the parallelization strategies for iterative reconstruction highlighted in Section 3 of the main text.
  - Key elements supporting forward projection include:
    - **`BasicFP.t`** implements core functionality for FP operations, either with or without parallel execution via OpenMP.
    - **`SerialFP.h`** (*sFP*) implements the serial FP operations that serve as our baseline FP metric for scaling performance.
    - **`ParallelFP.h`** (*pFP*) implements parallel FP operations using the OpenMP *parallel for* construct to exploit pixel-level parallelism over ray-sum computations.
    - **`pvmPrepassFP.h`** (*pvmFP*) implements parallel FP operations with distance computation for pixel spacing work group assignment using the OpenMP *parallel for* construct to exploit pixel-level parallelism and per-voxel mutexes to ensure correct updates.
    - **`vpBaseFP.h`** implements core functionality for parallel FP operations that use voxel projection to compute conflict-free task/thread mappings for subsequent parallel BP operations.
    - **`vpaPrepassFP.h`** (*vpaFP*) implements computations for conflict-free task/thread mappings by tracking the maximum projection extents across all voxels of the reconstruction volume.
    - **`vpePrepassFP.h`** (*vpeFP*) implements computations for conflict-free task/thread mappings by tracking the maximum projection extents across only edge voxels of the reconstruction volume.
  - Key elements supporting backprojection include:
    - **`BasicBP.t`** implements core functionality for BP operations, either with or without parallel execution via OpenMP.
    - **`SerialBP.h`** (*sBP*) implements the serial BP operations that serve as our baseline BP metric for scaling performance.
    - **`pvmParallelBP.h`** (*pvmBP*) implements parallel BP operations using the OpenMP *parallel for* construct to exploit pixel-level parallelism and per-voxel mutexes to ensure correct updates.
    - **`psParallelBP.h`** (*psBP*) implements parallel BP by assigning tasks to threads using the pixel spacing work group assignment data computed during FP together with the OpenMP *parallel for* construct to exploit pixel-level parallelism.
- **`apps/`** contains code implementing two utility applications and a driver program supporting our scalability study.
  - **`volume_gen.cc`** generates the *sinewave* synthetic XCT

phantom dataset according to various command line parameters.
  - **`simulate_xct.cc`** generates simulated XCT projections of an input volume according to various command line parameters.
  - **`art.cc`** executes iterative reconstruction using any one of the valid $FP + BP$ combinations described in Section 3.1 of the main text.

- Useful shell scripts include:
  - **`gen_vol.sh`** generates the complete set of *sinewave* synthetic XCT phantom datasets used in our scalability study.
  - **`gen_pct.sh`** generates the complete set of *sinewave* simulated XCT projections used in our scalability study.
  - **`run_bench.sh`** executes the full suite of experiments used in our scalability study.
  - **`cat_results.sh`** concatenates results of scaling tests into a single space-delimited text file for post-processing.

- Other helpful files include:
  - **`CMakeLists.txt`** provides content for compiling our prototype XCT reconstruction system using the CMake build system.
  - **`LICENSE`** provides information governing redistribution and use of source and binary forms of our system.
  - **`README`** provides instructions for building the code and running the driver program.

Additional Results

Recall from Section 3.2 of the main text that we execute our initial performance study using several systems with various hardware configurations:

- *Test Platform #0* (*TP0*)—a Debian 8.11 system with two Intel Xeon E5-2699 v3 2.30 GHz processors (36 cores, 72 hardware threads), 64 GB of RAM, and GCC 7.3.0.
- *Test Platform #1* (*TP1*)—an Ubuntu 18.04 system with two Intel Core i7-7820X 3.60 GHz processors (8 cores, 16 hardware threads), 64 GB of RAM, and GCC 7.4.0.
- *Test Platform #2* (*TP2*)—an Ubuntu 16.04 system with two Intel Core i7-7800X 3.50 GHz processors (6 cores, 12 hardware threads), 64 GB of RAM, and GCC 5.4.0.

In the main text, we include results for only *TP0*. Here, we include results for *TP1* in Figures 1–4 and for *TP2* in Figures 5—8. Generally speaking, we observe the same trends on these platforms as on *TP0*.

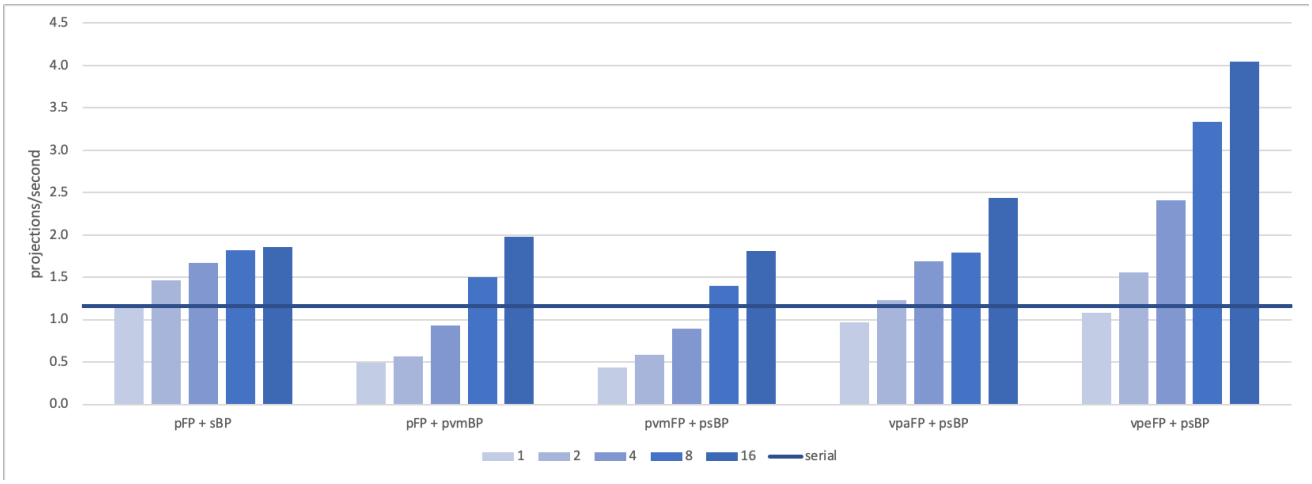

Figure 1: *TP1* - Reconstruction performance with 100^3-voxel reconstruction volume. *All five parallelization strategies outperform the serial baseline when using 8 or 16 threads, with* vpeFP + psBP *performing best overall; however, with far fewer threads than* TP0*, strategies employing per-voxel mutexes (*pFP + pvmBP *and* pvmFP + psBP*) simply add overhead and thus underperform the serial baseline at lower thread counts.*

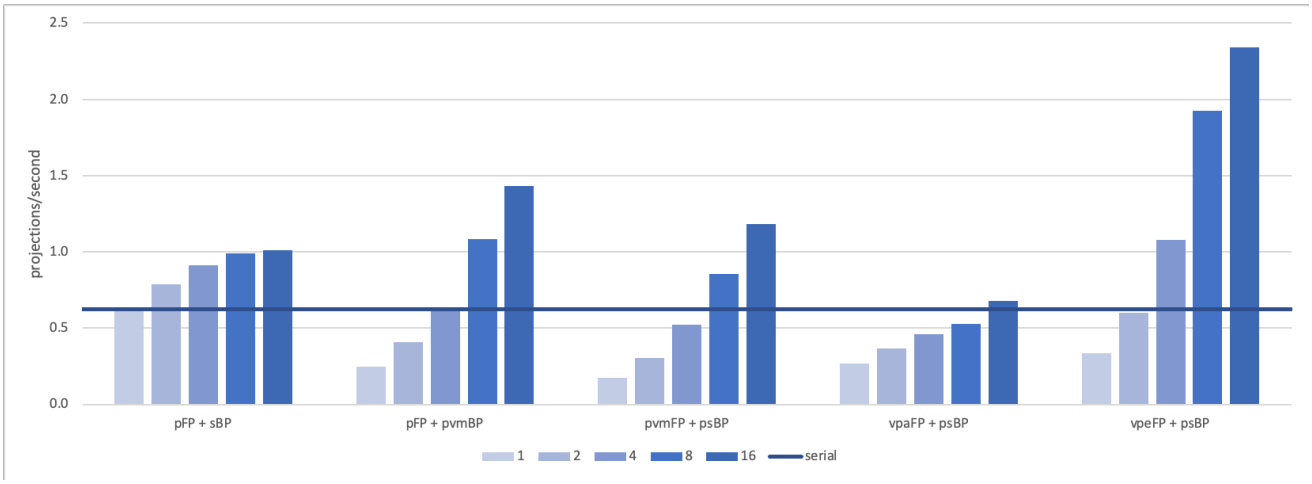

Figure 2: *TP1* - Reconstruction performance with 200^3-voxel reconstruction volume. *As in the 100^3-voxel case,* vpeFP + psBP *performs best overall, while projecting all voxels (*vpaFP + psBP*) is simply too costly for even this relatively low-resolution reconstruction volume. As with* TP0*,* pFP + pvmBP *also begins to show some promise for this 200^3-voxel reconstruction volume.*

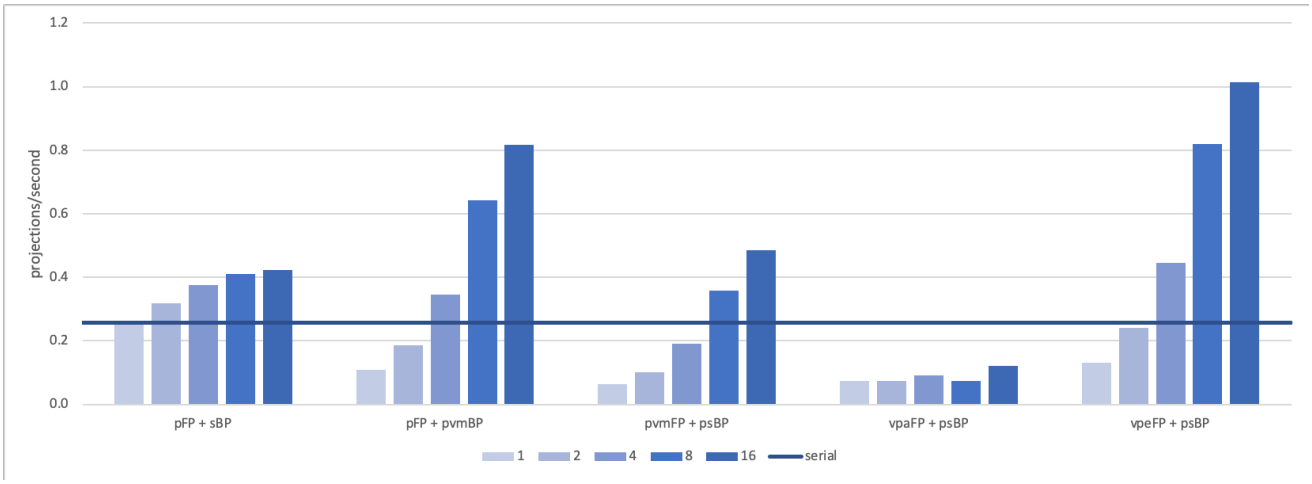

Figure 3: *TP1* - Reconstruction performance with 400^3-voxel reconstruction volume. *Here, too, projecting all voxels (*vpaFP + psBP*) performs poorly, while* pFP + pvmBP *becomes more attractive, underperforming* vpeFP + psBS *by only about 20% with 16 threads.*

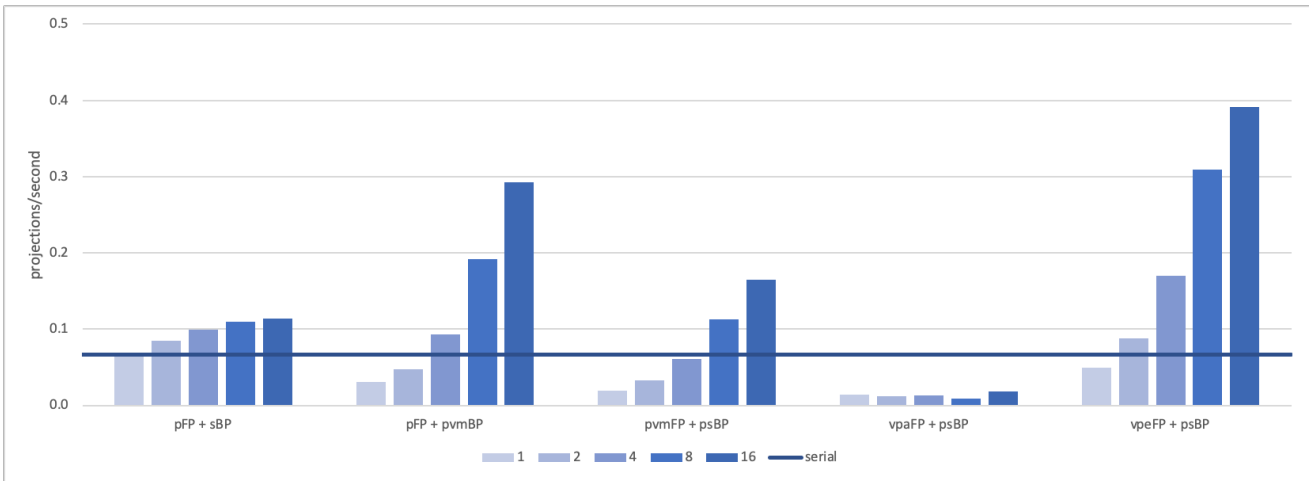

Figure 4: *TP1* - Reconstruction performance with 800^3-voxel reconstruction volume. *As with* TP0*, projecting all voxels (*vpaFP + psBP*) simply does not scale. In contrast,* vpeFP + psBP *performs best overall on this platform, but* pFP + pvmBP *remains viable in situations involving high-resolution reconstruction volumes.*

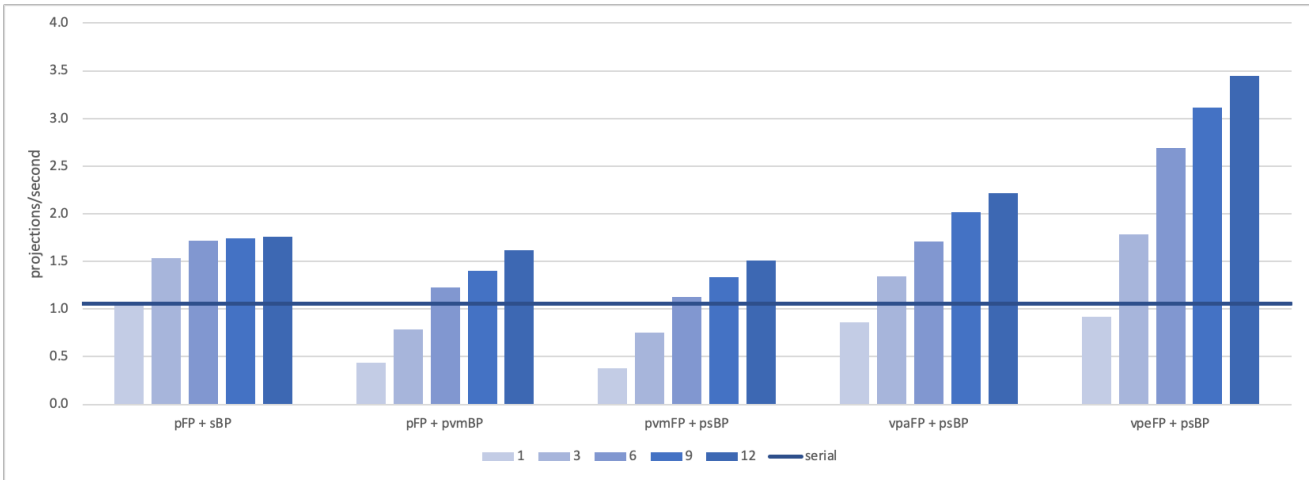

Figure 5: *TP2* - Reconstruction performance with 100^3-voxel reconstruction volume. *Here, all five parallelization strategies outperform the serial baseline when using 6–12 threads, with* vpeFP + psBP *performing best overall; however, as with* TP1*, strategies employing per-voxel mutexes (*pFP + pvmBP *and* pvmFP + psBP*) simply add overhead and thus underperform the serial baseline at lower thread counts.*

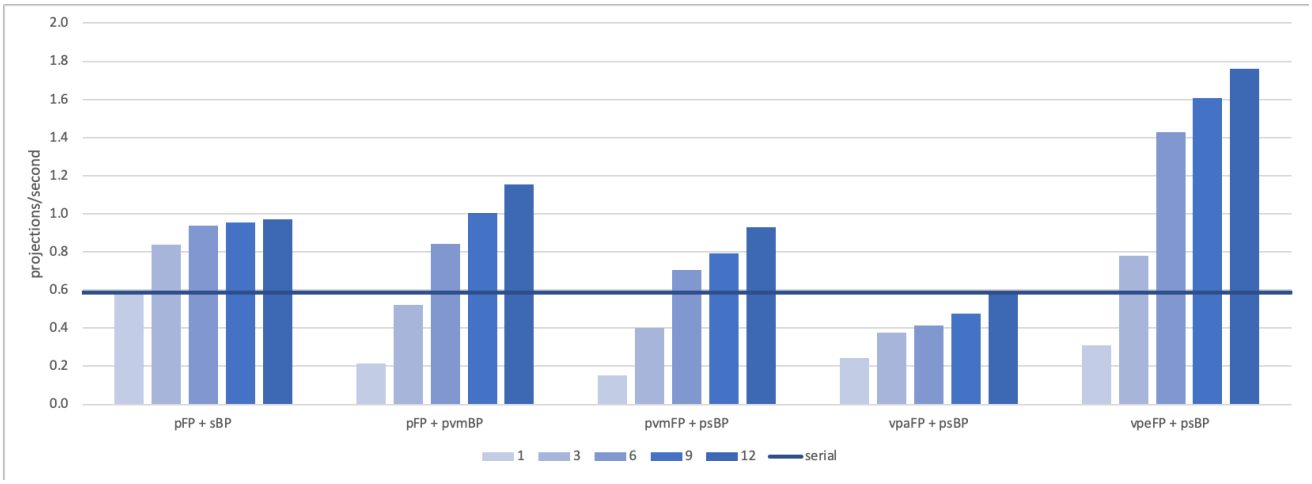

Figure 6: *TP2* - Reconstruction performance with 200^3-voxel reconstruction volume. *As in the 100^3-voxel case,* vpeFP + psBP *performs best overall, while projecting all voxels (*vpaFP + psBP*) is again too costly for even this relatively low-resolution reconstruction volume. As with* TP0 *and* TP1*,* pFP + pvmBP *also begins to show some promise for this 200^3-voxel reconstruction volume.*

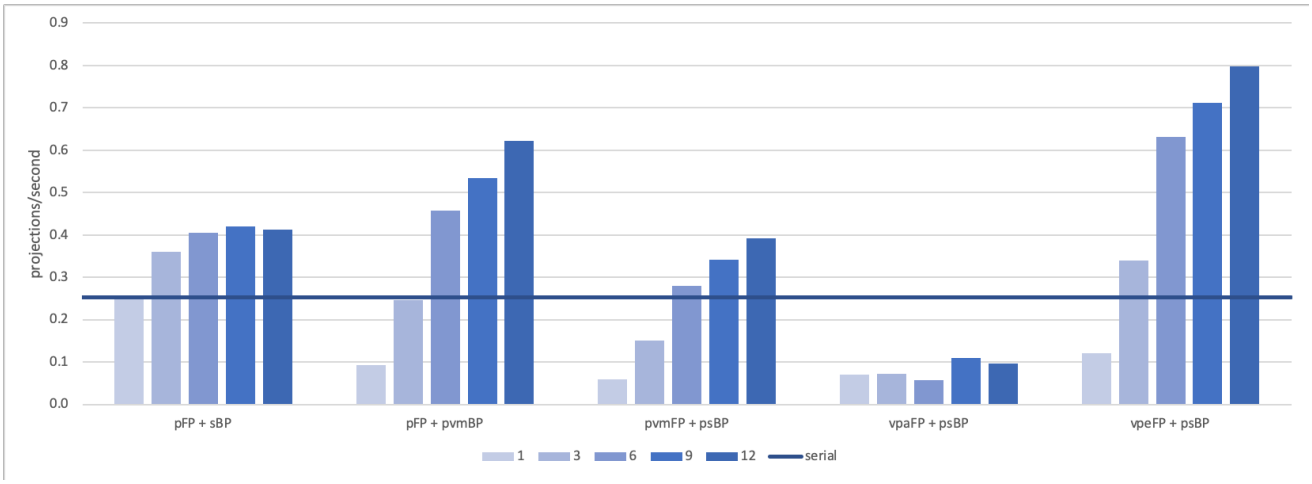

Figure 7: *TP2* - Reconstruction performance with 400^3-voxel reconstruction volume. *Projecting all voxels (*vpaFP + psBP*) continues to perform poorly, while* pFP + pvmBP *becomes more attractive, underperforming* vpeFP + psBS *by only about 25% with 12 threads.*

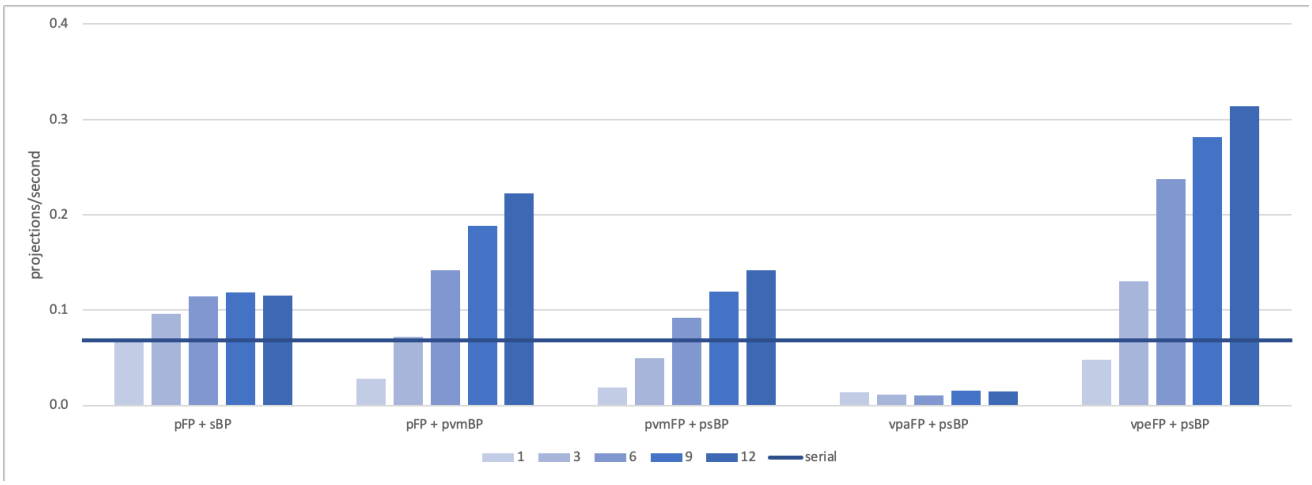

Figure 8: *TP2* - Reconstruction performance with 800^3-voxel reconstruction volume. *As with the lower-resolution volumes, projecting all voxels (*vpaFP + psBP*) simply does not scale,* vpeFP + psBP *performs best overall, and* pFP + pvmBP *remains viable in situations involving high-resolution reconstruction volumes.*