GPU acceleration of 3D forward and backward projection using separable footprints for X-ray CT image reconstruction

Meng Wu and Jeffrey A. Fessler

EECS Department
University of Michigan

Fully 3D Image Reconstruction Conference
Workshop on High-Performance Image Reconstruction (HPIR)

July 11, 2011

http://www.eecs.umich.edu/∼fessler

Overview

- Forward / back-projection is primary bottleneck for iterative reconstruction
- Tradeoff between computational complexity and system model accuracy
- Separable footprint approximation for cone-beam X-ray CT
- implementations
  - multi-core CPU (shared memory)
  - multi-GPU

Typical iteration for statistical image reconstruction in CT

Penalized weighted least-squares (PWLS) cost function:

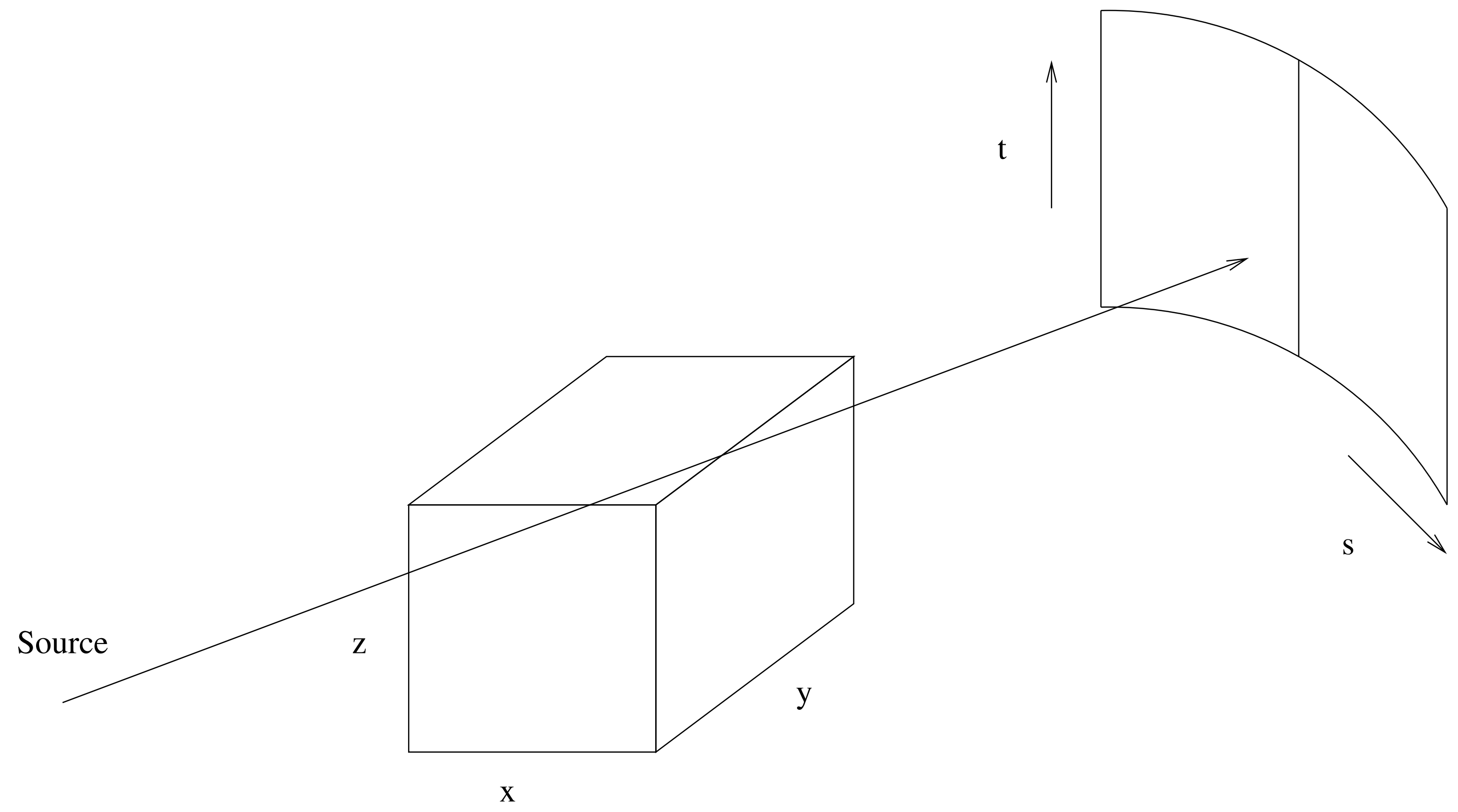
$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \Psi(\boldsymbol{x}), \;\; \Psi(\boldsymbol{x}) = \sum_{i=1}^{n_\mathrm{d}} \frac{w_i}{2} (y_i - [\boldsymbol{A}\boldsymbol{x}]_i)^2 + \mathsf{R}(\boldsymbol{x})$$

- unknown 3D image $\boldsymbol{x} = (x_1, \ldots, x_{n_\mathrm{p}})$ with n_p voxels
- $\boldsymbol{y} = (y_1, \ldots, y_{n_\mathrm{d}})$ CT (log) projection data with n_d rays
- w_i statistical weighting for ith ray, $i = 1, \ldots, n_\mathrm{d}$
- $\boldsymbol{A}$: $n_\mathrm{d} \times n_\mathrm{p}$ system matrix
- $\mathsf{R}(\boldsymbol{x})$: edge-preserving regularizer
- forward projector : $[\boldsymbol{A}\boldsymbol{x}]_i = \sum_{j=1}^{n_\mathrm{p}} a_{ij} x_j$.

OS-type iteration:

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \boldsymbol{D} \left(\boldsymbol{A}' \boldsymbol{W} \left(\boldsymbol{y} - \boldsymbol{A} \boldsymbol{x}^{(n)} \right) - \nabla \mathsf{R}\left(\boldsymbol{x}^{(n)} \right) \right)$$

Cone-beam geometry



- x, y, z image voxel coordinates
- s, t detector coordinates
- β source position
- Assume z and t axes are parallel

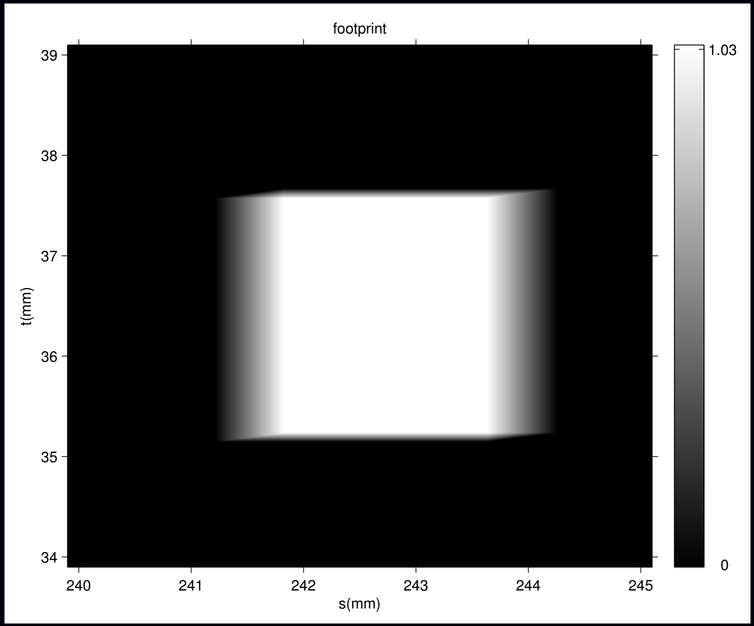
3D forward- / back- projectors for X-ray CT

Mathematically:

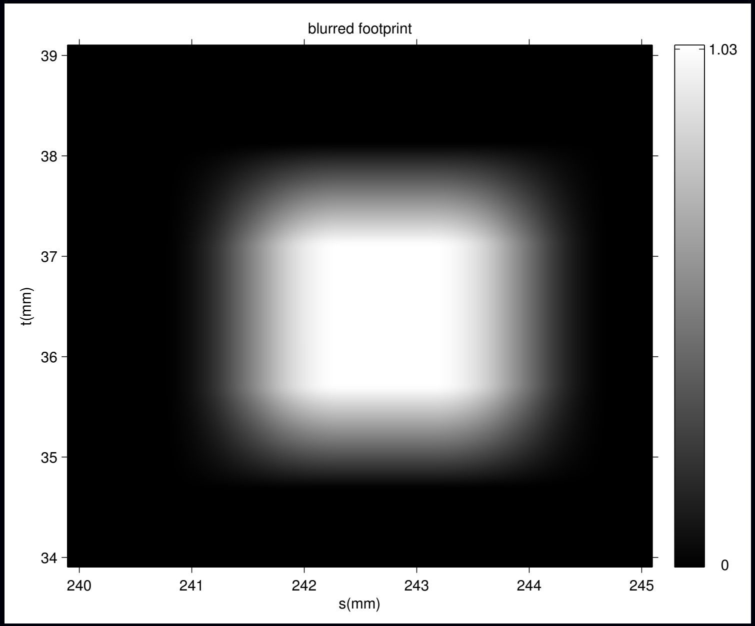
$$\begin{aligned} \text{3D forward projector: } g(s,t,\beta) &= \sum_{x,y,z} a(s,t,\beta;x,y,z) f(x,y,z) \\ \text{3D back-projector: } b(x,y,z) &= \sum_{s,t,\beta} a(s,t,\beta;x,y,z) g(s,t,\beta) \end{aligned} \qquad (1)$$

- $f(x,y,z)$: image voxel values at 3D spatial location x,y,z
- $g(s,t,\beta)$: measured projection views
- $a(s,t,\beta;x,y,z)$: system model that describes the footprints of the voxel centered at x,y,z blurred by the detector response
- Typical detector response corresponds to detector element size.

System model / voxel footprints



Line-integral footprint
$q(s,t,\beta;x,y,z)$



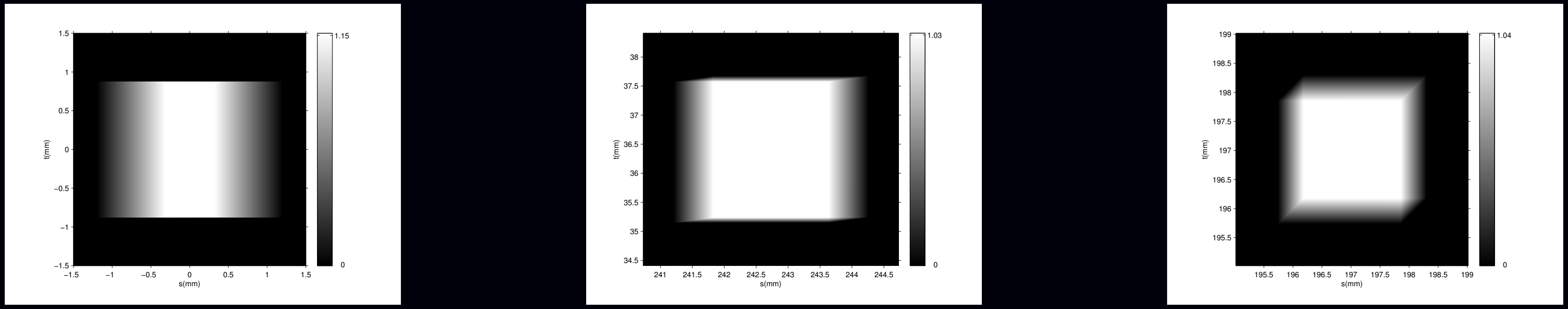
Blurred footprint (big!)
$a(s,t,\beta;x,y,z)$

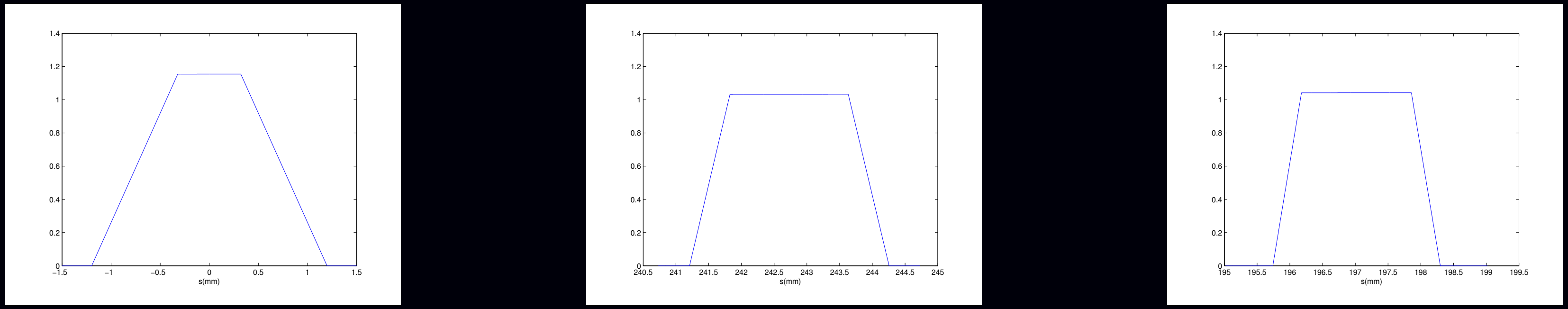
Shift-invariant detector response $h(s,t)$:

$$a(s,t,\beta;x,y,z) = \iint h(s-s',t-t')\, q(s',t',\beta;x,y,z)\, \mathrm{d}s'\, \mathrm{d}t' \, .$$

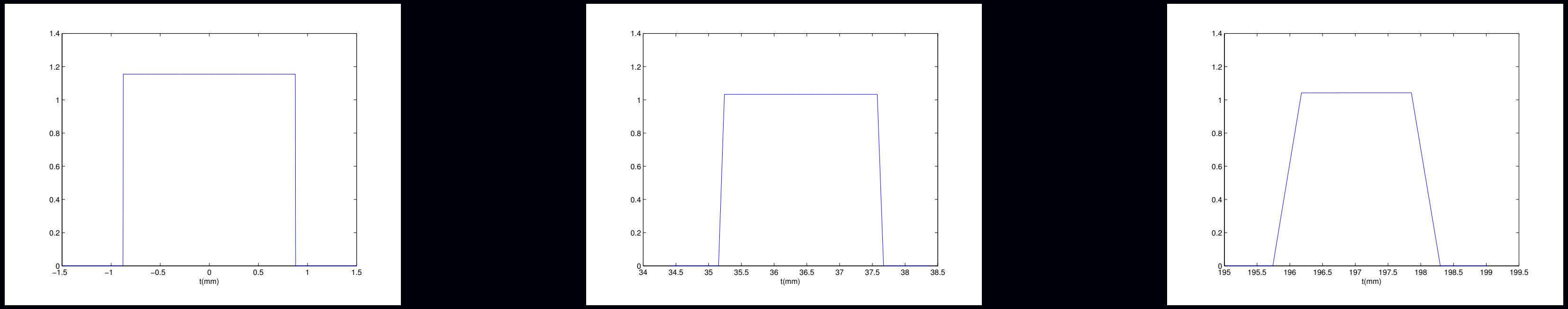
Rectangular detector elements: $h(s,t) = \frac{1}{r_s r_t} \underbrace{\operatorname{rect}\left(\frac{s}{r_s}\right) \operatorname{rect}\left(\frac{t}{r_t}\right)}_{\text{separable}} .$

Line-integral footprints





Profiles in s (transaxial)



Profiles in t (axial)

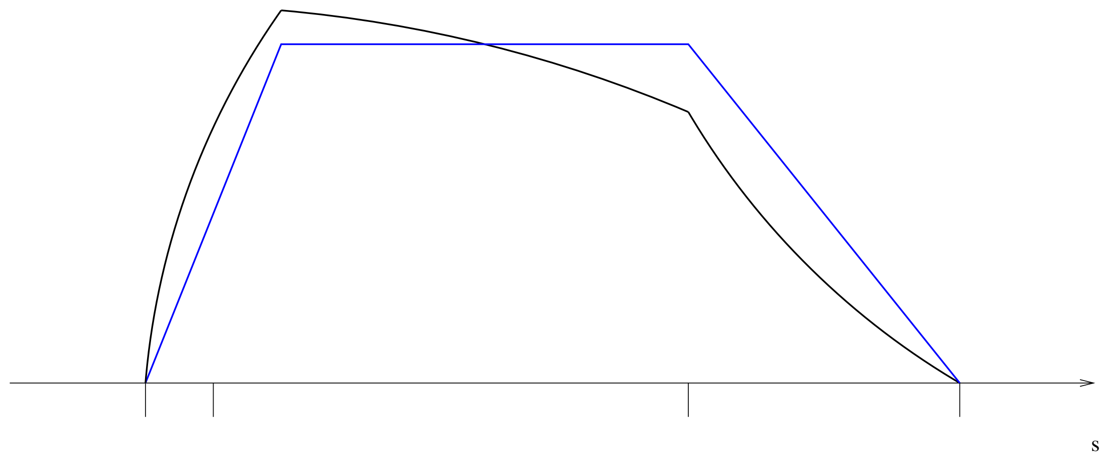
Separable footprint (SF) approach: Approximation 1

SF approximation for line-integral footprint function: (Yong *et al.*, T-MI, 2010):

$$q(s,t,\beta;x,y,z) \triangleq v(s,t,\beta)\,u(\beta;x,y)\,\tilde{F}_1(s,\beta;x,y)\,\tilde{F}_2(t,\beta;x,y,z).$$

- $\tilde{F}_1$: trapezoidal footprint function for transaxial direction (along det. row)
- $\tilde{F}_2$: rectangular footprint function for axial direction (along det. column)
- $u(\beta;x,y)$: voxel-dependent amplitude function
- $v(s,t,\beta)$: ray-dependent amplitude function.
  (The amplitude functions require minimal computation time.)

Trapezoid vertices match exactly the projections of voxel boundaries.



Separable footprint (SF) approach: Approximation 2

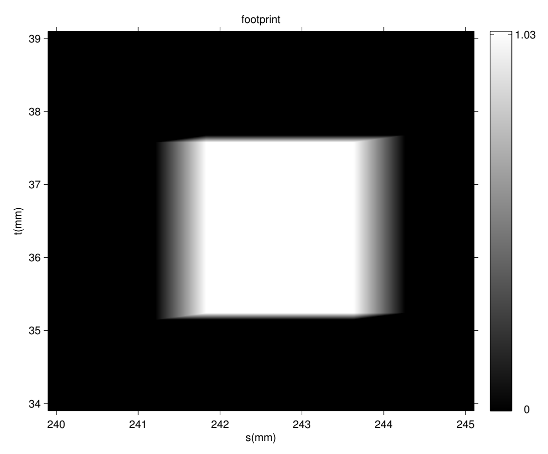
Combining with separable model for detector blur $h(s,t)$ yields
SF approximation for (blurred) footprint function:

$$a(s,t,\beta;x,y,z) \triangleq v(s,t,\beta)\, u(\beta;x,y)\, F_1(s,\beta;x,y)\, F_2(t,\beta;x,y,z). \qquad (2)$$

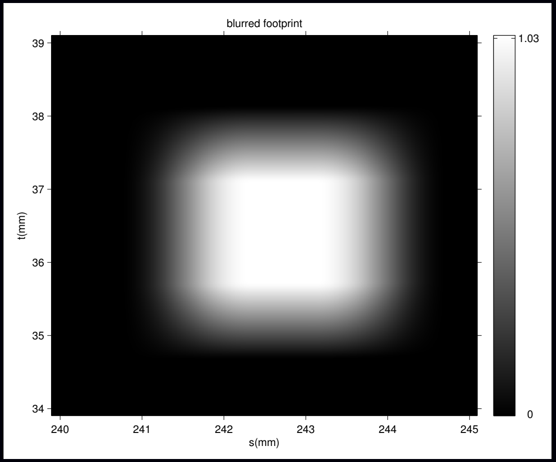
- $F_1(s) \triangleq \frac{1}{r_s} \operatorname{rect}\left(\frac{s}{r_s}\right) * \tilde{F}_1(s), \qquad F_2(t) \triangleq \frac{1}{r_t} \operatorname{rect}\left(\frac{t}{r_t}\right) * \tilde{F}_2(t)$

- Closely approximates the true blurred footprint for small cone angles.
- More accurate than distance-driven (DD) approximation.
(Self-consistent across resolution scales.)
- The main computational work is related to F_1 and F_2.
- Separability simplifies implementation.

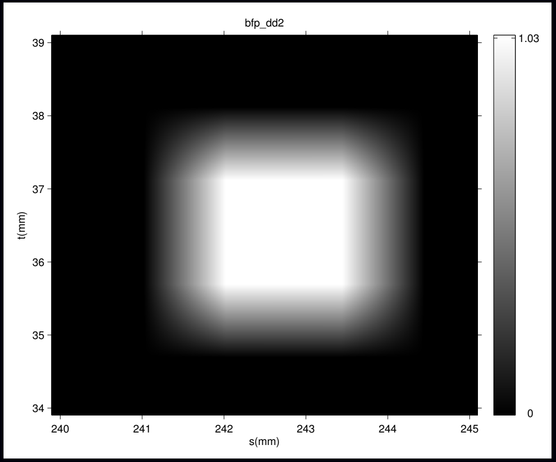
Blurred footprint approximations: near $z = 0$



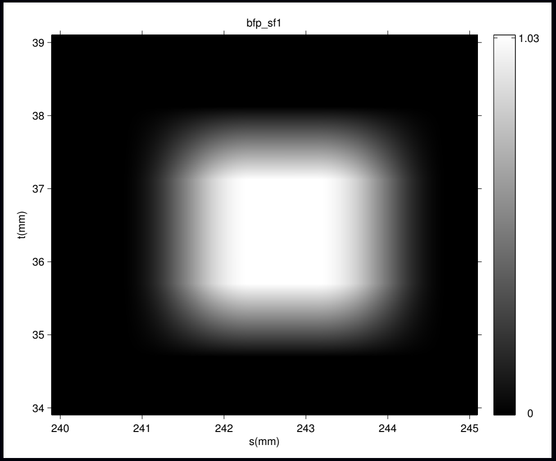
$q(s,t;\beta;\vec{n})$



$q(s,t;\beta;\vec{n})$



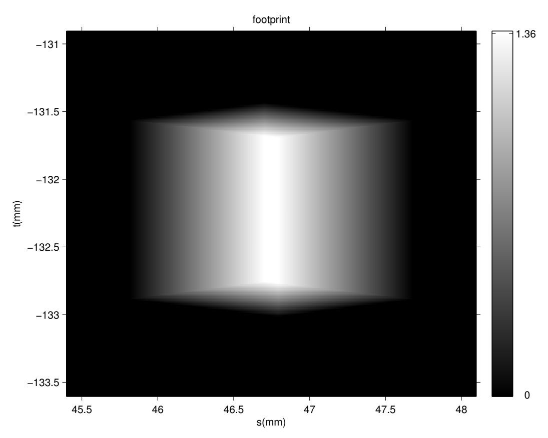
DD



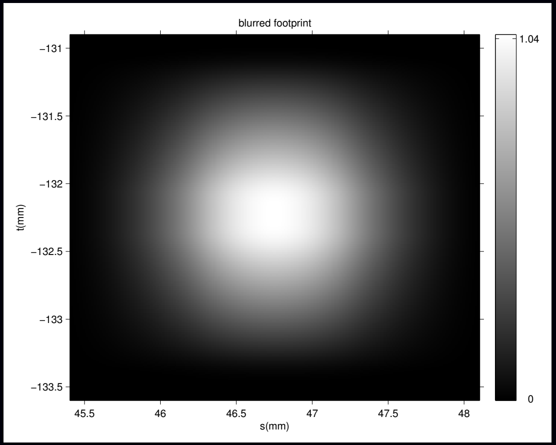
SF

$\vec{n} = (100, 150, 15)$, $\beta = 0$, Azimuthal angle through voxel center: 14.3°. Polar angle: 2.1°.

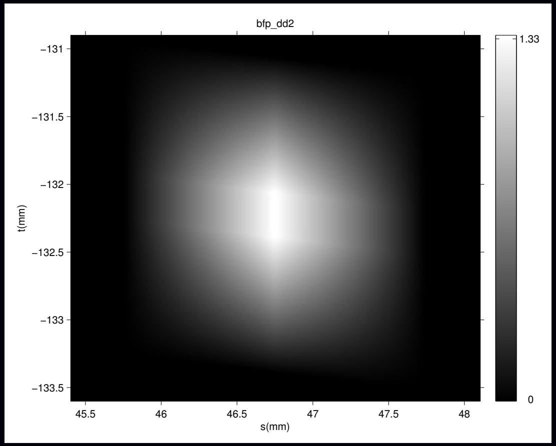
Blurred footprint approximations: off center



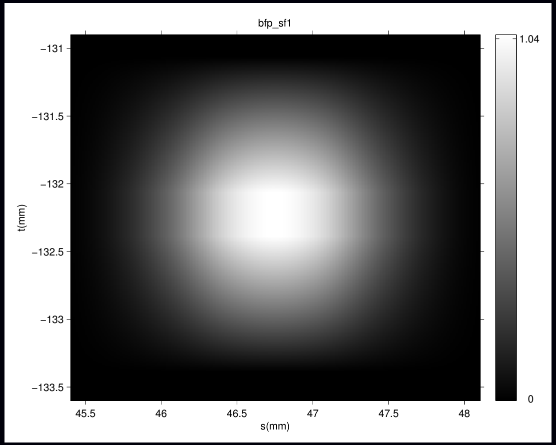
$q(s,t;\beta;\vec{n})$



$q(s,t;\beta;\vec{n})$



DD



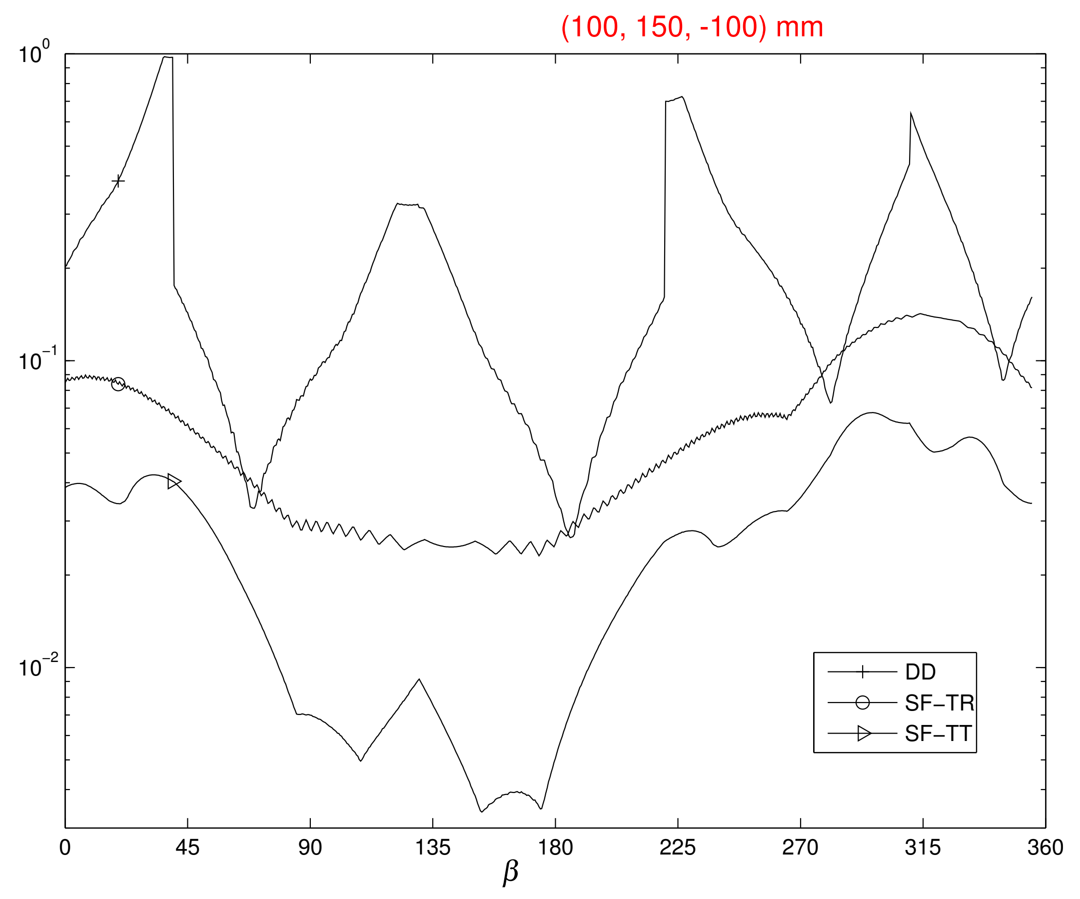
SF

$\vec{n} = (100, 150, -100)$. $\beta = 135°$. Azimuthal angle through voxel center: 138°. Polar angle: 7.8°.

SF projector maximum error (single voxel)

Worst-case error between exact blurred footprint and an approximation:

$$e(\beta;\vec{n}) \triangleq \max_{s,t\in\mathbb{R}} \left|F(s,t;\beta;\vec{n}) - F_{\text{approximation}}(s,t;\beta;\vec{n})\right|$$



Maximum errors on a logarithmic scale for a 1mm^3 size voxel.

Long *et al.*, IEEE T-MI, Nov. 2010.

SF implementation

Efficient implementation of forward projection (1) using separability (2):

$$\begin{aligned} g(s,t,\beta) &= \sum_{x,y,z} a(s,t,\beta;x,y,z) f(x,y,z) \\ &= v(s,t,\beta) \sum_{x,y} F_1'(s,\beta;x,y) \underbrace{\left[\sum_z F_2(t,\beta;x,y,z) f(x,y,z) \right]}_{z \text{ inner loop (axial)}}, \end{aligned} \tag{3}$$

for modified transaxial footprint function:

$$F_1'(s,\beta;x,y) \triangleq u(\beta;x,y) F_1(s,\beta;x,y). \tag{4}$$

Back-projector:

$$b(x,y,z) = \sum_\beta \sum_t F_2(t,\beta;x,y,z) \underbrace{\left[\sum_s F_1'(s,\beta;x,y)\, g'(s,t,\beta) \right]}_{s \text{ inner loop (transaxial)}},$$

view-dependent scaling of projection views:

$$g'(s,t,\beta) \triangleq v(s,t,\beta)\, g(s,t,\beta).$$

SF projector implementation on multi-core CPU

$$g(s,t,\beta) = \underbrace{v(s,t,\beta)}_{\text{scale}} \underbrace{\sum_{x,y} F_1'(s,\beta;x,y)}_{\text{loop}} \underbrace{\left[\sum_z F_2(t,\beta;x,y,z) f(x,y,z)\right]}_{\text{unroll}}$$

- each core/thread handles a distinct set of projection views
  - for each β in set
    - initialize working projection view array to zero: $g'(s,t;\beta) := 0$
    - for each x,y
      - compute $F_1'(s,\beta;x,y)$ (trapezoid $*$ rect; typically 2-10 samples)
      - for each t ($\sum_z$ is 1-3 terms so unroll):

$$p(t;x,y,\beta) := \sum_z F_2(t,\beta;x,y,z) f(x,y,z)$$

      - accumulate into working projection view array

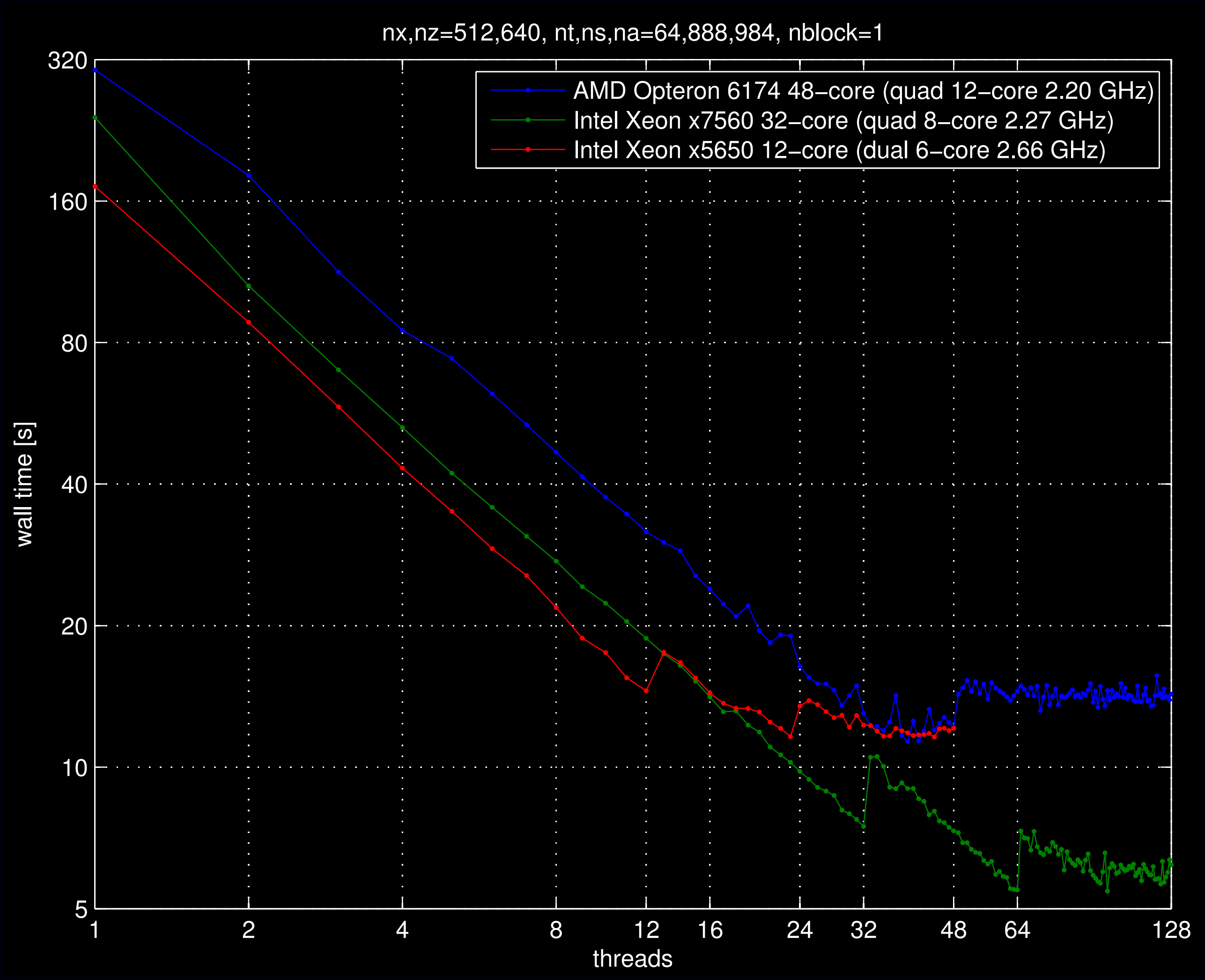
$$g'(s,t;\beta) \mathrel{+}= F_1'(s,\beta;x,y)\, p(t;x,y,\beta)$$

    - scale view and write to main memory:

$$g(s,t,\beta) := v(s,t,\beta) g'(s,t;\beta).$$

Parallelization by view is easy on multi-core CPU,
but speedup can be limited by memory bandwidth.

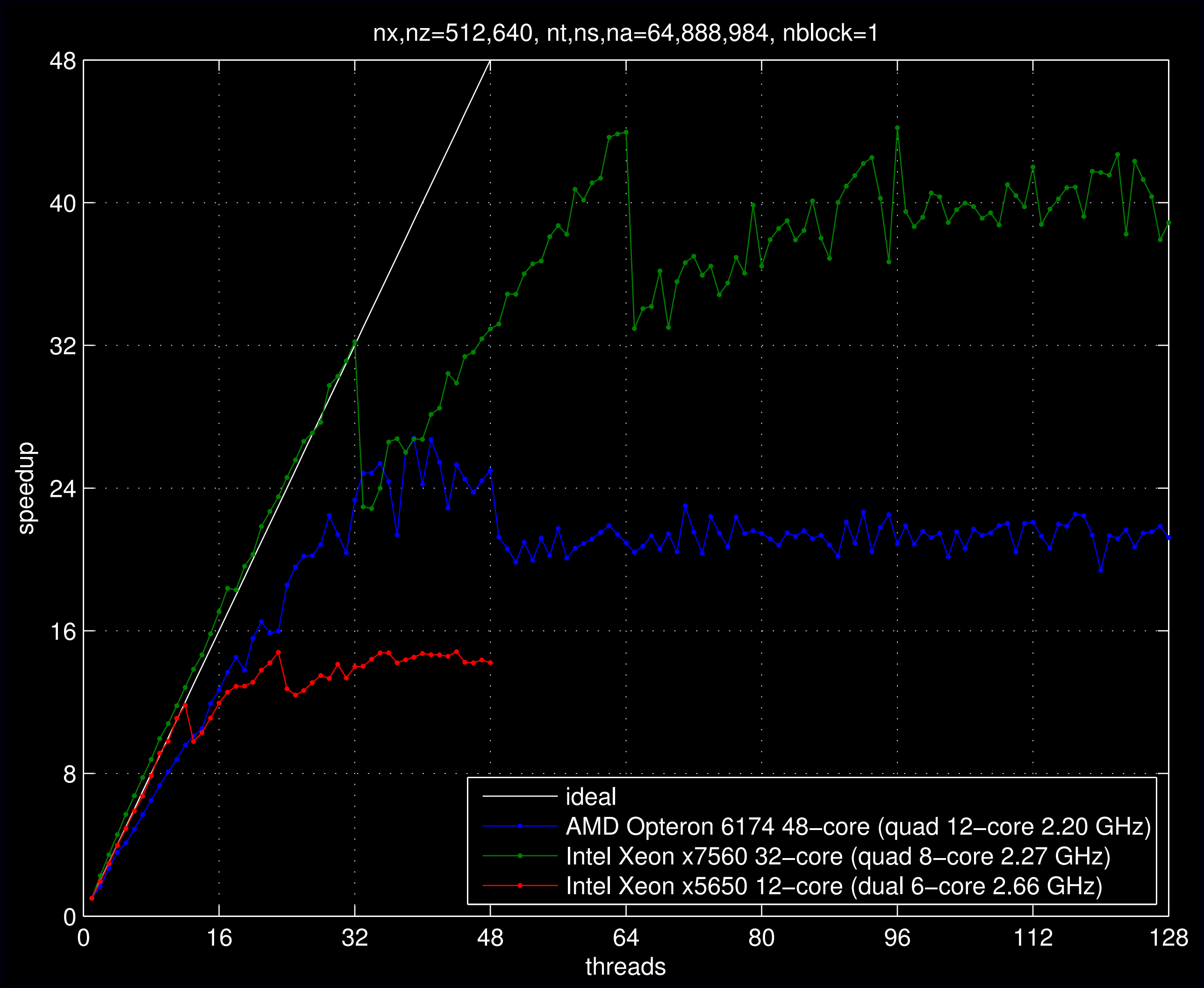
SF projector implementation on CPU: Wall time



64-slice CT, one helix turn.

Shortest time: 5.5 sec for 64 (or 96) threads on Intel 32-core.

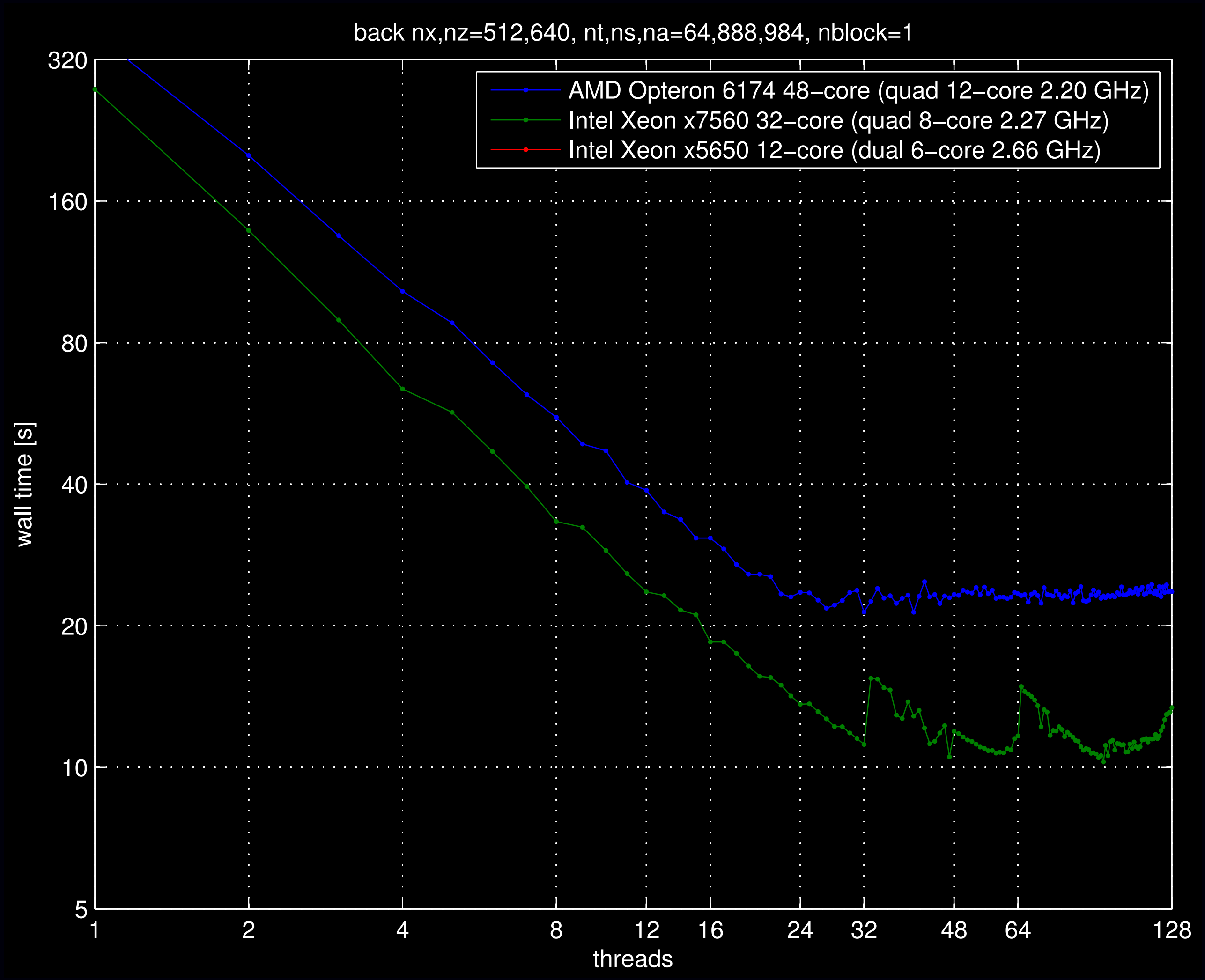
SF projector implementation on CPU: Speedup



Speedup saturates at about 20× on 48-core AMD Opteron system, with parallelization across views.

Memory bandwidth even more of an issue on GPU $\Longrightarrow$ *not* view based.

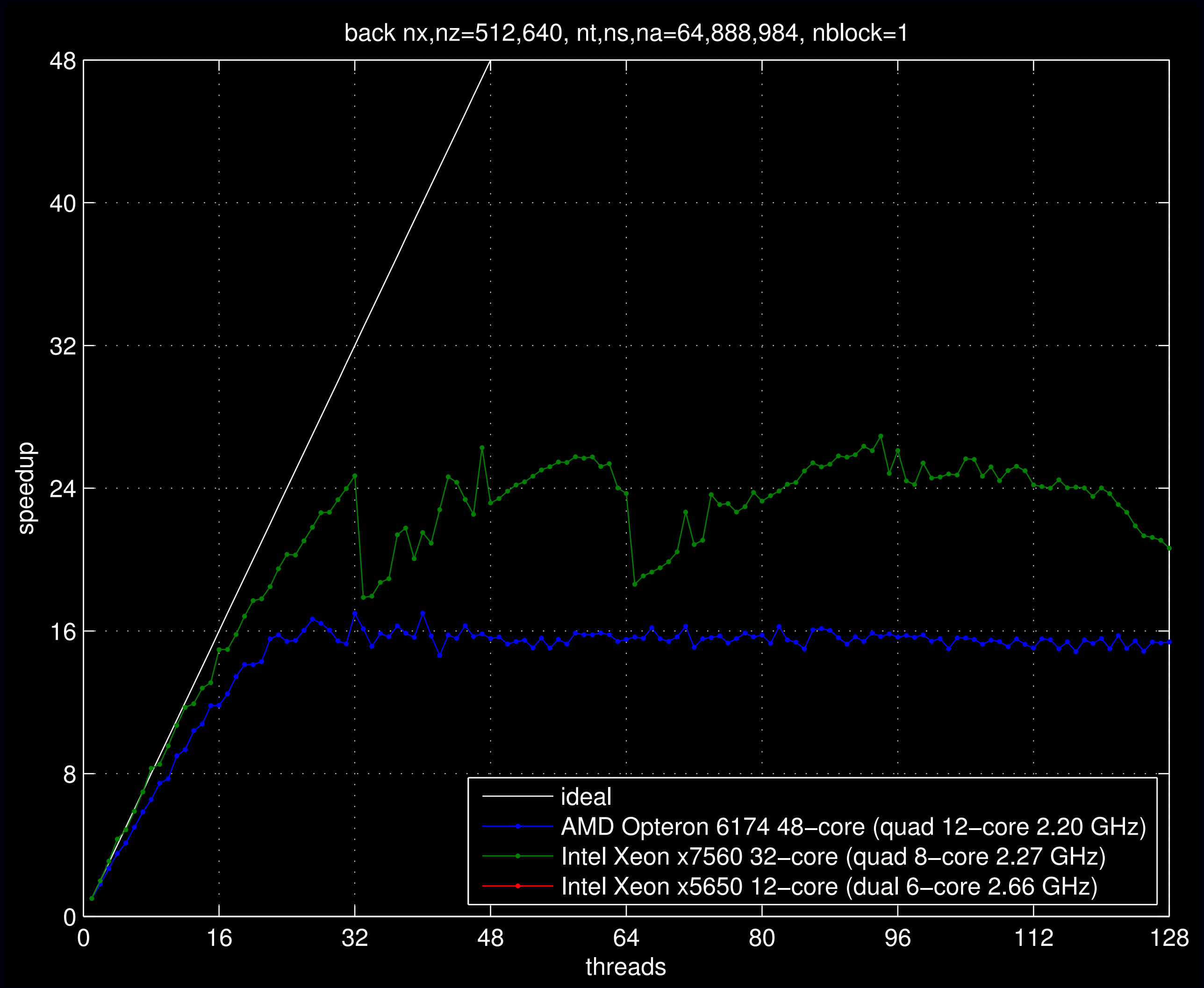
SF back-projector implementation on CPU: Wall time



64-slice CT, one helix turn.

Shortest time: 10.3 sec for 94 threads on Intel 32-core.

SF back-projector implementation on CPU: Speedup



Speedup saturates at about 16× on 48-core AMD Opteron system, with parallelization across x,y locations.

SF implementation on GPU: Version 1

$$g(s,t,\beta) = v(s,t,\beta) \sum_{x,y} F_1'(s,\beta;x,y) \left[\sum_z F_2(t,\beta;x,y,z) f(x,y,z)\right]$$

For each view: (All threads work on a single view.)

- $p(t,x,y;\beta) := 0$ [64,512,512]
- Kernel 1
  - parfor each x,y: compute and store $F_1'(s,\beta;x,y)$ [10,512,512]
- Kernel 2
  - parfor each x,y,z: $p(t,x,y;\beta)$ += $F_2(t,\beta;x,y,z) f(x,y,z)$ [64,512,512]

    (Typically each voxel contributes to at most 3 detector rows, *i.e.*, values of t.)
- Kernel 3
  - parfor each x,y and t: $g'(s,t;\beta)$ += $F_1'(s,\beta;x,y)\, p(t,x,y;\beta)$, [888,64]

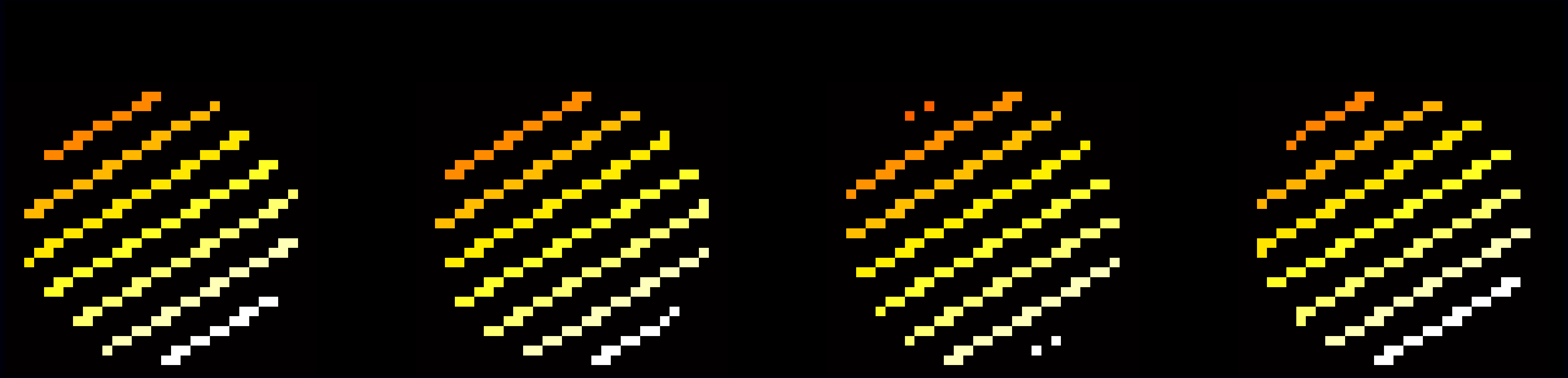
    (Typically each voxel contributes to 2-10 detector columns, *i.e.*, values of s.)
- Kernel 4
  - parfor each s and t: $g(s,t,\beta) = v(s,t,\beta) g'(s,t;\beta)$

- More intermediate storage than CPU version.
- Read-modify-write errors. Synchronization impractical.

Disjoint footprint decomposition

Footprints of different voxels overlap $\Longrightarrow$
parallelization across arbitrary (x, y) values causes read-modify-write errors.

Solution: identify sets of voxel strips having *disjoint footprints*.



Set 1 Set 2 Set 3 Set 4

- Loop over sets
- Parallelize across strips.
- Loop over voxels within strips

Parallelization with disjoint footprints

- Typical number of transaxial detector channels: $N_{\mathrm{s}} = 888$
- Typical largest footprint size: 10
- $\Longrightarrow$ 88-way parallelization within each voxel set

Further parallelization:

- Across detector rows $N_{\mathrm{t}} = 64$
- Across helix turns for same angles
- Across projection views (if memory / bandwidth permits)

Details

voxel set: $\mathscr{V}_s \triangleq \{(x,y) : s_{\min}(\beta, x, y) = s\}$.

- $\mathscr{V}_0, \mathscr{V}_{10}, \mathscr{V}_{20}, \ldots$
- $\mathscr{V}_1, \mathscr{V}_{11}, \mathscr{V}_{21}, \ldots$
- $\vdots$
- $\mathscr{V}_9, \mathscr{V}_{19}, \mathscr{V}_{29}, \ldots$

SF implementation on GPU: Version 4

$$g(s,t,\beta) = v(s,t,\beta) \sum_{x,y} F_1'(s,\beta;x,y) \left[\sum_z F_2(t,\beta;x,y,z) f(x,y,z)\right]$$

For each view β: (All threads work on a single view.)

- $g'(s,t;\beta) := 0$ [888,64]
- GPU Kernel 1
  - parfor each x,y: compute and store $F_1'(s,\beta;x,y)$ [10,512,512]
- CPU function 2: form voxel set lists $\mathscr{V}_s$ for this view β [888,1024]
- CPU loop: for k=0:9
  - GPU Kernel 3
    - parfor each t and $s' \in \{s : \mathrm{mod}(s,10) = k\}$:
      - $p(s,t,s';k,\beta) := 0$ [10,64,88]
      - loop over x,y in $\mathscr{V}_{s'}$:

        $$p(s,t,s';k,\beta) \mathrel{+}= F_1'(s,\beta;x,y) \sum_z F_2(t,\beta;x,y,z) f(x,y,z)$$
      - accumulate: $g'(s,t;\beta) \mathrel{+}= p(s,t,s';k,\beta)$
  - GPU Kernel 4
    - parfor each s and t: $g(s,t,\beta) = v(s,t,\beta) g'(s,t;\beta)$

(Parallelization across helix turns and views omitted for simplicity.)

GPU vs CPU results

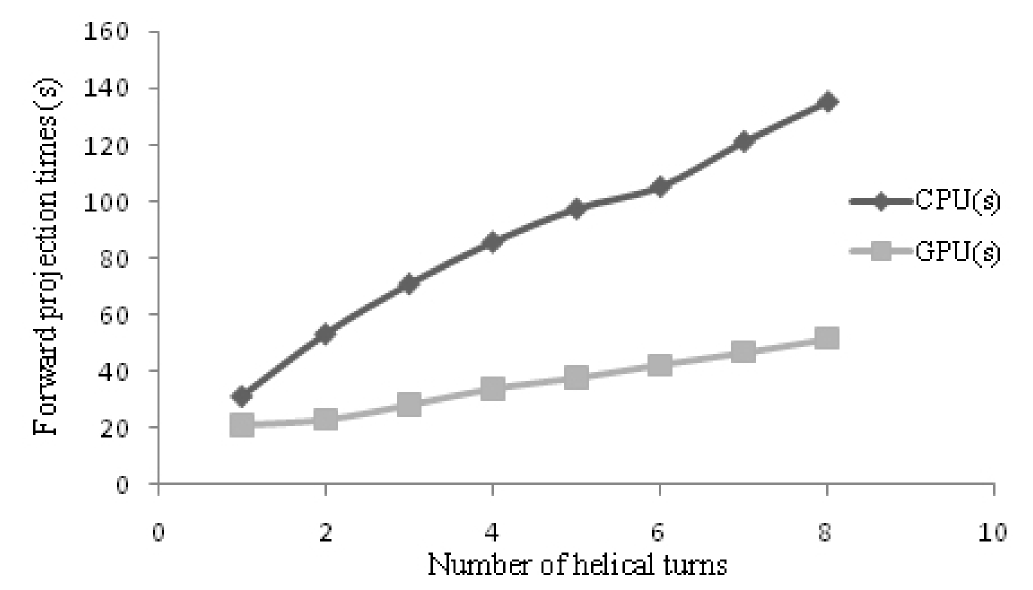
- NVDIA CUDA
- Tedious manual optimization of number of blocks/threads
- GE LightSpeed X-ray CT geometry: $N_{\mathrm{s}} = 888$ detector channels, $N_{\mathrm{t}} = 64$ detector rows, $N_{\beta} = 984$ views over 360°.
- 3D object size: $512 \times 512 \times 640$.
- Times averaged over 5 runs.
- CPU: 12-core (two 2.66 GHz Intel Xeon X5650 processors); 24 threads
- GPU: four 1.15 GHz NVDIA Tesla C2050

Single helix turn results

Forward projection computation time for 1 helical turn, single GPU.

GPU kernel 1	CPU function 2	GPU kernel 3	GPU kernel 4	Total
7.8 s	9.9 s	2.1 s	1.1 s	20.9 s

Footprint F_1' and voxel set construction dominates execution time.
Footprints and voxel sets can be re-used across helix turns.
Only GPU kernel 3 and 4 are needed for subsequent turns.



GPU memory use

- 640 MB for 3D image
- 2 MB for 8 projection views
- 10 MB for F_1', F_2, u, v etc.

Total less than 1 GB.
Tesla C2050 has 3 GB.

Global memory accesses:

- $12 N_{\mathrm{x}} N_{\mathrm{y}}$ in kernel 1,
- $22 N_{\mathrm{s}} N_{\mathrm{t}}$ in kernels 3 and 4.

Results for 8-turn helix: Multiple GPUs

Multiple GPU parallelization

- forward projection: distribute views
- back projection: partition x, y plane

Forward and back-projection computation times
24-thread CPU version and GPU version for 8-turn helix

	CPU	single GPU	dual GPU	quad GPU	when
Forward projection	145 s	52 s	45 s	45 s	abstract
Back projection	156 s	114 s	71 s	50 s	abstract
	100 s	44 s	26 s	33 s	new

Dual-GPU version is 3-4 × faster than 12-core (24-thread) CPU version, for 8-turn helix.

Limited by memory bandwidth? Lack of experience?
More investigation needed...

Summary

- Separable footprint method amenable to parallelization with multi-core CPU or GPU
- CPU parallelization across views was trivial. $42\times$ speedup over single CPU-core on expensive 32-core system
- GPU implementation provided modest acceleration factors with substantial programming pain.*
- Perhaps (much?) greater acceleration possible with further optimization.

- Tesla C2050: $2400
- Intel Xeon x7560: $4000

Possible improvements to CPU version based on GPU experience

- Use disjoint voxel sets so that multiple cores work on same view?
- Exploit symmetry between helix turns (re-using footprints and voxel sets).

$*$ perhaps more so for the student than for the professor...

Bibliography

[1] Y. Long, J. A. Fessler, and J. M. Balter. 3D forward and back-projection for X-ray CT using separable footprints. *IEEE Trans. Med. Imag.*, 29(11):1839–50, November 2010.