INTER*CTIVE supercomputing

Hierarchical Parallelization of a Radio Frequency Tomography Application via Multiple GPUs

Dana Schaa, Northeastern U.

Mark Barnell, AFRL/Rome

Roope Astala, Viral Shah, Niraj Srivastava, and Steve Reinhardt, Interactive Supercomputing

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Agenda

What we set out to do
What we did
What's left to do
What we learned

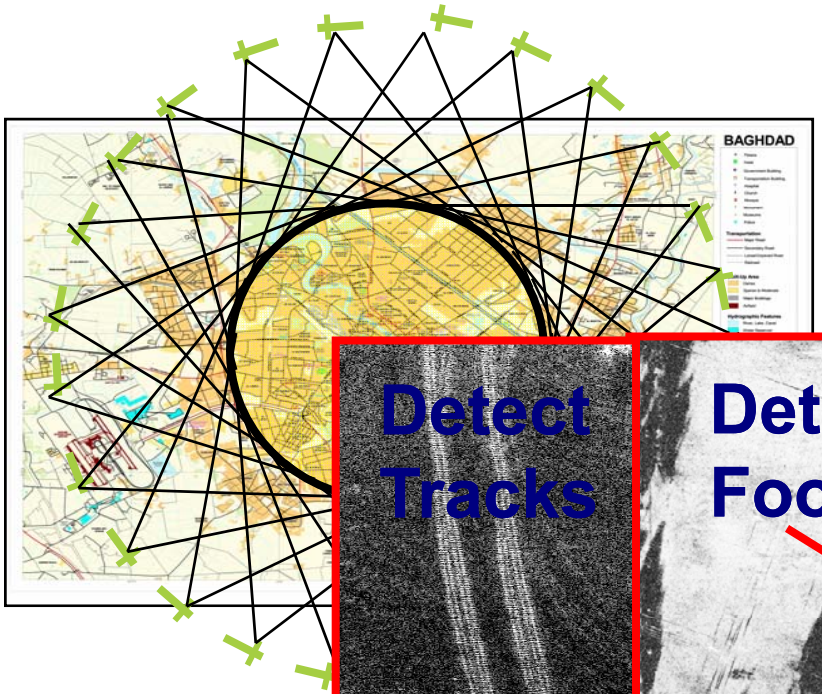
© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

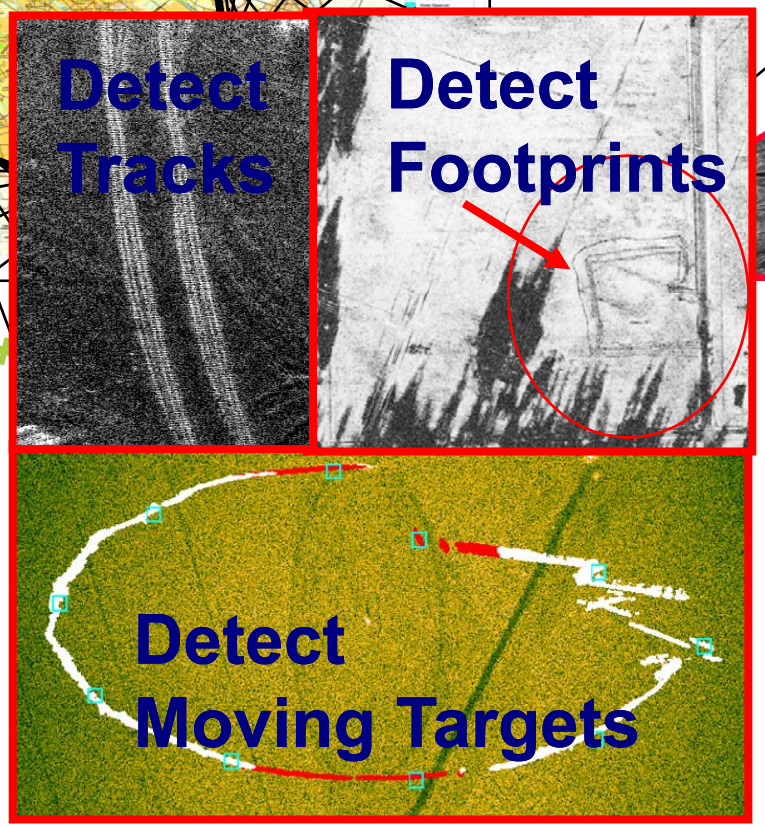
Agenda

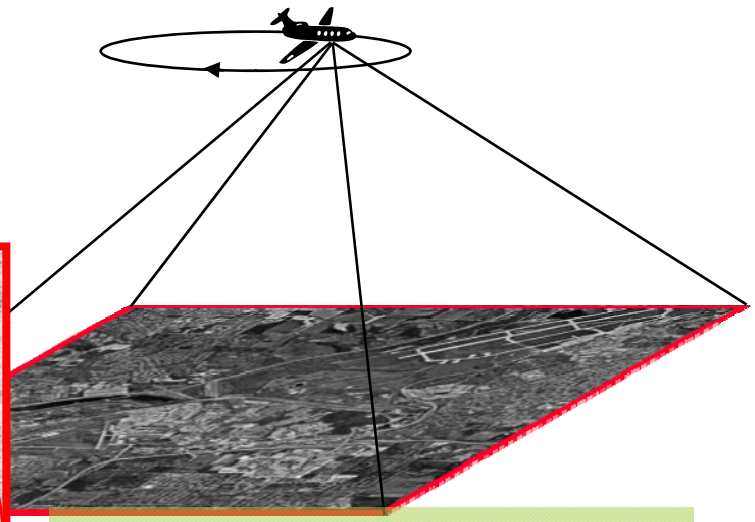
- What we set out to do
- What we did
- What's left to do
- What we learned

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Radio Frequency Tomography





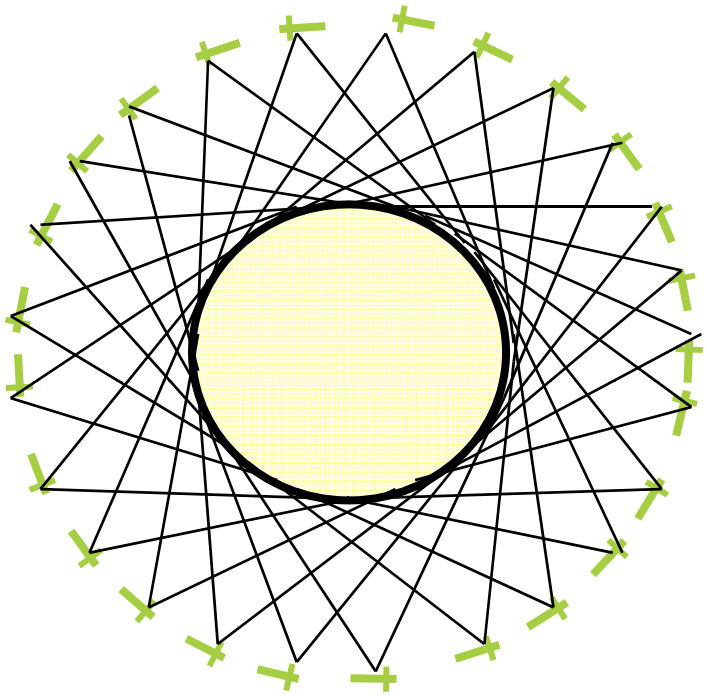


Staring Sensing

Key points:
- critically important for certain military tasks
- computationally demanding, esp. when large data used
- how to run it radically faster (and cheaply and power-efficiently)?

© 2009, Interactive Supercomputing, Inc.

Parallelism



The contribution of each pulse can be calculated with loop parallelism

Within each pulse, abundant data parallelism exists

→ Exploit the data parallelism within a GPU

→ Exploit the loop parallelism between GPUs

Do this as much as possible in the M language to enable rapid algorithmic development

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

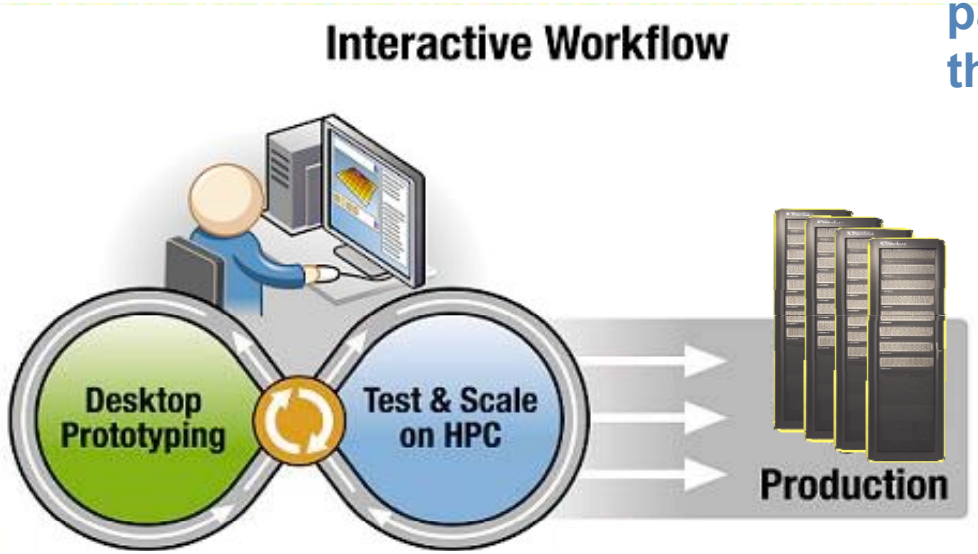
Agenda

What we set out to do
What we did
What's left to do
What we learned

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Star-P Bridges Scientists to HPCs

Star-P enables domain experts to use parallel, big-memory systems via the M language of MATLAB® and other desktop languages




```
% explicitly parallel with *p
n=10000*p

% implicitly parallel
A = rand(n, n);

% implicitly parallel
x = randn(n, 1);

% implicitly parallel
y = zeros(size(x));

while norm(x-y) / norm(x) > 1e-11
    y = x;
    x = A*x;
    x = x / norm(x);
end;
```

M

MATLAB® is a registered trademark of The MathWorks, Inc. ISC's products are not sponsored or endorsed by The Mathworks, Inc.
© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

INTERACTIVE supercomputing

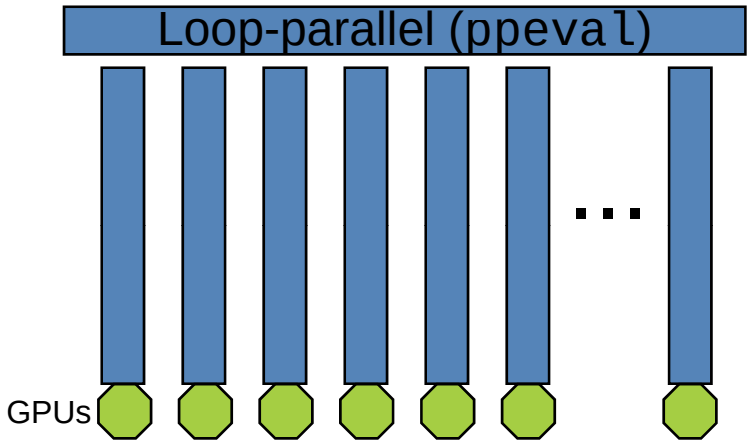
Linkage with CUDA Kernel

Use Star-P's loop-parallel construct to decompose the data and control iterations

Call the CUDA kernel (via Mex) from each iteration

Appropriate for

- **highly compute-intensive ...**
- **loop-parallel constructs (*i.e.*, no communication)**
- **data movement to/from GPU after operation**



MATLAB® is a registered trademark of The MathWorks, Inc. ISC's products are not sponsored or endorsed by The Mathworks, Inc.

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Code Changes

AFRL/Rome code

- Most SLOC are M, but 99+% of the time is spent in a C kernel

Key kernel

- Already in C, ported to CUDA
- Refactored to be callable on a subset of the full data

Higher-level parallelism done in M/Star-P

- For-loop becomes parallel For-loop

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Productivity

C -> CUDA: 34 hours

- Porting and optimization
- Note: by very sharp grad student with 3 years' CUDA experience in imaging applications
- Small kernel: ~100 SLOC

C -> M rewrite: 10 hours

- To be able to understand algorithm quickly and make further changes easily
- ~30 SLOC

CUDA-Star-P linkage: ~4 hours

Tuning and running on various configurations: ~16 hours

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Systems

Bale GPGPU/Viz cluster at Ohio Supercomputer Center

- 18 nodes, each with 2 AMD 2.6 GHz dual-core Opteron CPU sockets, 8GB of RAM, and 2 NVIDIA Quadro FX 5600 Graphics cards, with PCI-e 16X links

Lincoln cluster at National Center for Supercomputing Applications

- 96 nodes (only used up to 48 for these experiments, to eliminate any bandwidth-sharing effects)
- Each a Dell PowerEdge 1950 server (2 quad-core Harpertown 2.33GHz sockets and 16GB of RAM), with server pairs sharing access to a Tesla S1070 unit via PCI-e Gen2 8X links

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Performance

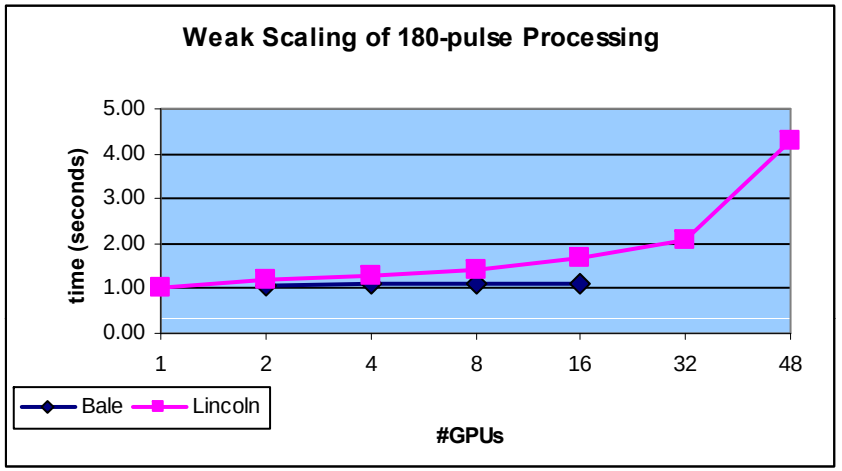
180 pulses nominal data size (and multiples)

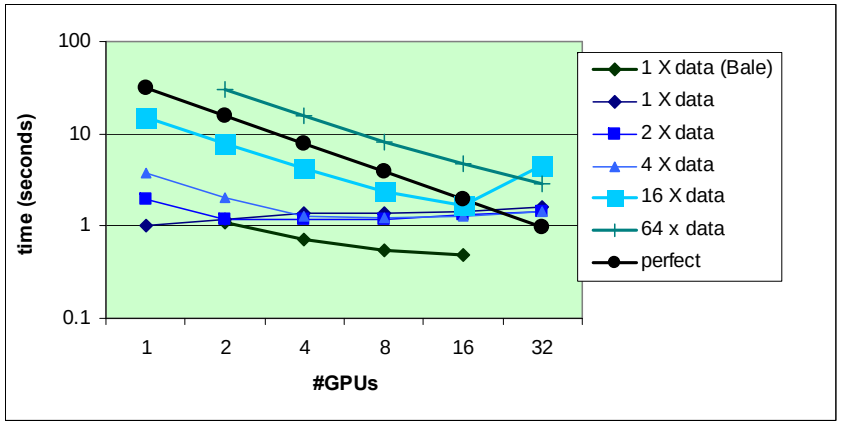
Weak scaling

- (Single-core x86-64 45.20s)
- 1 node (2 GPUs) 1.07s
- Bale: 8 nodes (16 GPUs) (8x more data) 1.12s
- Lincoln: degrades noticeably above 8 GPUs; may be due to PCI-e 8X links

Strong scaling

- (note log-log scale)
- Good scaling down to ~1 second (the granularity of 180 pulses running on 1 GPU)
- Fundamental connectivity and bandwidth issues still pertain





Note: 32 GPUs ~= 8,000 cores

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

INTERACTIVE supercomputing

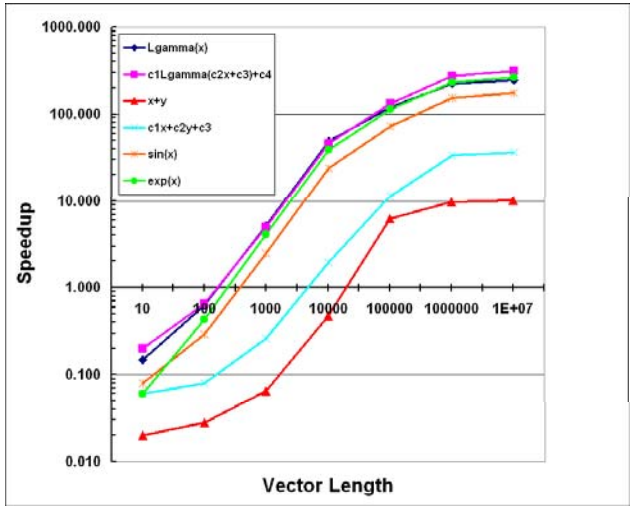
Agenda

What we set out to do
What we did
What's left to do
What we learned

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Explore optimum chunk size for hierarchical parallelism

- Balance using all available GPUs with the minimum practical sizes for specific devices



Current pseudocode

```
for i=1:length(x)/180
    for j=1:180   %pulses
        process_pulse();
    end
end
```

Possible pseudocode

```
nGPU =          % #GPUs in use
chunk_min =     % empirically calculated
if length(x)/180 < nGPU
  chunk_sz = chunk_min;
else
  chunk sz = 180;
end
nchunks = length(x)/chunk_sz;
for i=1:nchunks
    for j=1:chunk_sz   %pulses
        process_pulse();
    end
end
```

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Agenda

What we set out to do
What we did
What's left to do
What we learned

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Lessons

CUDA<->Star-P linkage straightforward

- O(1 staff-day) to implement

Weak scaling performance very good

Strong scaling performance good to a point, needs more exploration of finer granularity

Underlying bandwidth/connectivity must be sufficient to support the distribution of work

M language is suitable for expression of high-level parallelism across multiple GPUs

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

INTER*CTIVE supercomputing

For further info, contact Steve Reinhardt at

sreinhardt atatat InteractiveSupercomputing.com

Acknowledgments: Use of GPU-enabled clusters at the Ohio Supercomputer Center (Bale) and the National Center for Supercomputing Applications (Lincoln)

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

INTERACTIVE supercomputing

Back-up

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Wrapper code

```
/*
 * phd   - input only (everyone gets all data)
 * image - output only (everyone writes to entire array, must be summed
later)
 * paux  - input only (everyone gets all data)
 * X1    - input only (everyone calculates the same.  j
 * Y1    - input only
 */
//void gpu_backproject(complex_type *phd, complex_split image, double *paux,
//                     FloatType *X1, FloatType *Y1)

//int main(int argc, char *argv[])
void mexFunction(int nlhs, void *plhs[],
                 int nrhs, void *prhs[])
//------------------------------------
{

     // CREATE SOME DUMMY OUTPUT
     //mxArray* dummy = prhs[0];
     //plhs = mxDuplicateArray(dummy);

    int ii, jj;

    FILE * fid;
    double *paux;

    // RA: Get the output block size from mex inputs
    int nxmin;
    int nxmax;
    nxmin = (int)mxGetScalar(prhs[3]);
    nxmax = (int)mxGetScalar(prhs[4]);

    //Initialize some values
    int nXobs = N;
    int nYobs = N;
    int nZobs = N;

    FloatType *X1, *Y1;

    ThreadInfo threadInfo[NUMTHREADS];
    pthread_t threads[NUMTHREADS];

#ifdef TIMER
    double elapsed = getTime();
#endif

    fid = fopen(SPECFILE_OUT, "w");
    fprintf(fid, "dirOut %s\n", outputdir);
    fprintf(fid, "fileBase %s\n", filebase);
    fprintf(fid, "x1 %d\n", xc - x_extent / 2);
    fprintf(fid, "x2 %d\n", xc + x_extent / 2);
    fprintf(fid, "y1 %d\n", yc - y_extent / 2);
    fprintf(fid, "y2 %d\n", yc + y_extent / 2);

    if (NPulses)
    {
        fprintf(fid, "startPulse %d\n", startPulse);
        fprintf(fid, "stopPulse %d\n", startPulse + NPulses);
    }

    fprintf(fid, "concatenateOutput\n");
    fprintf(fid, "outputFileExtension %s\n", backname);
    fclose(fid);

    //Create the necessary file names with the paths
    sprintf(phname, "%s/%s.%s", outputdir, filebase, backname);
    sprintf(pauxname, "%s/%s.paux.%s", outputdir, filebase, backname);
    sprintf(phheadername, "%s/%s.phxhdr.%s", outputdir, filebase, backname);
    sprintf(pngname, "%s/%s.image.%s.png", RESULT_DIR, filebase, backname);

#ifdef VERBOSE
    printf("InputData is %s\n", indata);
    printf("resultdir is %s\n", RESULT_DIR);
    printf("image out is %s\n", IMAGE_OUT);
    printf("phname=%s\n", phname);
    printf("pauxname=%s\n", pauxname);
    printf("phheadername=%s\n", phheadername);
    printf("pngname=%s\n", pngname);
#endif

    printf("Performing Digital Spotlight\n");

   //RA: Grab paux from mex array
   paux = (double *)mxGetPr(prhs[0]);
   printf("paux size is: %lu bytes\n", sizeof(double) * samples * N);

    // Now do backprojection
    printf("Performing Backprojection\n");
//    complex_split *images[NUMTHREADS];
    complex_split image;
    complex_type * phd;
```

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

Wrapper code (2)

```
    // Allocate space for the data
    printf("PHD size is: %lu bytes\n", sizeof(complex_type) * eff_ny * nxin);
    phd = (complex_type *) malloc(sizeof(complex_type) * eff_ny * nxin);
    if (phd == NULL) mallocErr(__FILE__, __LINE__);

    // RA: Somehow iscMex doesn't allow a pointer copy, let's do a deep copy

     FloatType* tmp_real = (FloatType*)mxGetPr(prhs[1]);
     FloatType* tmp_imag = (FloatType*)mxGetPr(prhs[2]);
     for(ii=0;ii<eff_ny*nxin;ii++)
                 {
                      phd[ii].real = tmp_real[ii];
                      phd[ii].imag = tmp_imag[ii];
                 }

    //Read the data
    // looks like dats is storesd as interleaved and is being read into
seperate buffers for matlab
    //int fsize = fread(phd, sizeof(complex_type), eff_ny * nxin, fid);
    //fclose(fid);

    /* Check dimensions */
    if (nXobs != nxin)
    {
        printf("Xobs has wrong size.\n");
        return;
    }

    if (nYobs != nxin)
    {
        printf("Yobs has wrong size.\n");
        return;
    }

    if (nZobs != nxin)
    {
        printf("Zobs has wrong size.\n");
        return;
    }

    /* Allocate output space for image */
    image.real = (FloatType *) calloc(nyout * nxout, sizeof(FloatType));
    if (image.real == NULL) mallocErr(__FILE__, __LINE__);

    image.imag = (FloatType *) calloc(nyout * nxout, sizeof(FloatType));
    if (image.imag == NULL) mallocErr(__FILE__, __LINE__);

    X1 = (FloatType *) malloc(sizeof(FloatType) * nxout);
    if (X1 == NULL) mallocErr(__FILE__, __LINE__);
```

```
     Y1 = (FloatType *) malloc(sizeof(FloatType) * nyout);
        if (Y1 == NULL) mallocErr(__FILE__, __LINE__);

        /* defines axes for output */
        /* this is not exactly right when eff_ny%2=1 */
    #define F_NXOUT 1024.0
    #define F_NXOUT_MINUS1 (F_NXOUT-1.0)
    #define F_NYOUT 1024.0
    #define F_NYOUT_MINUS1 (F_NYOUT-1.0)
        float x=0.0;

        FloatType axis[4] = {1.0,-1.0,0.0,0.0}; // will be image location in meters

        axis[0] = -100.0;
        axis[1] = 100.0;
        axis[2] = -100.0;
        axis[3] = 100.0;

         for (ii = 0; ii < nxout; ii++)
         {
             X1[ii] = axis[0] + (axis[1] - axis[0]) *
                      (x / (FloatType) F_NXOUT_MINUS1);
             Y1[ii] = axis[2] + (axis[3]-axis[2]) * (x / F_NYOUT_MINUS1);
             ++x;
         }
    #if VERBOSE //
    ************************************************************************
        printf("size = %d\n", size);
        printf("nxin = %d\n", nxin);
        printf("nxout = %d\n", nxout);
        printf("nyout = %d\n", nyout);
        printf("eff_ny = %d\n", eff_ny);
        printf("F_EFF_NY = %.10f\n", F_EFF_NY);
    //    printf("c4df = %.10f\n", c4df);
    //    printf("f4pic = %.10f\n", f4pic);
    #endif  //
    ************************************************************************
        // CALL GPU HERE
        for(ii = 0; ii < NUMTHREADS; ii++)
        {
           int iters_per_thread;

     // RA EDIT: Do the output in blocks, for ppeval-level parallelism
     //      iters_per_thread = (int)ceil((double)nxout/(double)NUMTHREADS);
           iters_per_thread = (int)ceil((double)(nxmax-nxmin)/(double)NUMTHREADS);
           printf("iters_per_thread = %d\n", iters_per_thread);

           if(ii != NUMTHREADS - 1)
           {
              threadInfo[ii].start = nxmin+iters_per_thread * ii;
```

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.

INTERACTIVE supercomputing

Wrapper code (3)

```
            threadInfo[ii].end = nxmin+iters_per_thread * (ii + 1);
        }
        else
        {
            threadInfo[ii].start = nxmin+iters_per_thread * ii;
            threadInfo[ii].end = nxmax;
        }
        printf("thread %d: start = %d, end = %d\n", ii, threadInfo[ii].start,
threadInfo[ii].end);
        threadInfo[ii].X1 = X1;
        threadInfo[ii].Y1 = Y1;
        threadInfo[ii].paux = paux;
        threadInfo[ii].phd = phd;
        threadInfo[ii].image = &image;
        threadInfo[ii].tid = ii;

        // phd is only about ~2MB for 180 pulses (just broadcast to everyone)
        // image may have deterministic indices, or we can just collect a full
        //    image from everyone and have *P add the vectors together
        // paux
//        gpu_backproject(phd, paux, X1, Y1);
        pthread_create(&threads[ii], NULL, gpu_backproject, (void*)&threadInfo[ii]);
    }

    for(ii = 0; ii < NUMTHREADS; ii++) {
pthread_join(threads[ii], NULL);
    }
#ifdef TIMER
    ELAPSED(elapsed);
    printf("Total Time:  %e seconds\n", elapsed);
#endif
     plhs[0] = mxCreateDoubleMatrix(nyout,nxmax-nxmin,mxREAL);
     plhs[1] = mxCreateDoubleMatrix(nyout,nxmax-nxmin,mxREAL);
     plhs[2] = mxCreateDoubleMatrix(nxout,1,mxREAL);
     plhs[3] = mxCreateDoubleMatrix(nxout,1,mxREAL);
     double *imagetmp_real = mxGetPr(plhs[0]);
     double *imagetmp_imag = mxGetPr(plhs[1]);
     double *X1tmp = mxGetPr(plhs[2]);
     double *Y1tmp = mxGetPr(plhs[3]);
     for(ii=nxmin*nyout;ii<nxmax*nyout;ii++)
     {
      imagetmp_real[ii-nxmin*nyout] = (double) image.real[ii];
      imagetmp_imag[ii-nxmin*nyout] = (double) image.imag[ii];
     }
     for(ii=0;ii<nxout;ii++)
     {
                 X1tmp[ii] = (double) X1[ii];
                 Y1tmp[ii] = (double) Y1[ii];
     }
    free(phd);
 //   return 0;
}
```

© 2009, Interactive Supercomputing, Inc. and its licensors. All rights reserved.