MIC-GPU: High-Performance Computing for Medical Imaging on Programmable Graphics Hardware (GPUs)

Code Optimization Case Study

Klaus Mueller, Ziyi Zheng, Eric Papenhausen

Stony Brook University

Computer Science

Stony Brook, NY

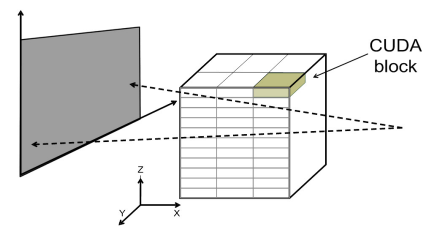
Optimization Case Study

Goal:

- test various optimization strategies and tweak to maximize impact
- using Filtered Backprojection for this case study

Optimization #1: minimize shared memory usage

- update a block of voxels per thread (optimum was 16 × 16 × 4)
- orientation-neutral block minimizes "shadow" on projections



Optimization Case Study

Optimization #2: exploit special GPU (ASIC) hardware

- we store projection data in texture memory
- allows fast bilinear interpolation
- frees up registers without penalty since texture is cached

Optimization #3: exploit constant memory

- we store projection (system) matrix in constant memory
- frees up shared memory and reduces global memory accesses

Optimization #4: increase thread granularity

- backproject multiple projections in one thread (optimum was 4)
- reduces global memory accesses and number of kernel invocations

Optimization Case Study

Optimization #5: Pre-fetching

- pre-fetch data while computing on previous data
- incurs some shared memory overhead but worked out OK

Optimization #6: Page-locked memory

- page-lock the result array
- forces OS to store this data on one contiguous page of memory
- eliminates the need for page swaps

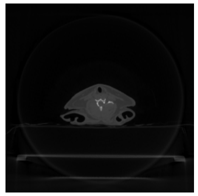
Other optimization strategies: loop unrolling, fast math

- we tried these but they did not yield much benefits in this specific case

Results

Platform

- NVIDIA GTX 480 GPU, CUDA 3.0 runtime API
- host: Intel Core 2 Duo CPU @ 2.66GHz



Data: 496 projections (1248×960)

- rabbit (from the RabbitCT website)
- reconstructed two volume sizes: 256^3, 512^3

RUN TIMES OF THE THREE CONFIGURATIONS

Configuration	Resolution	Total	Mean	Error	Speed-Up
Naïve	256^2	7.77 s	15.66 ms	8.04 HU^2	N/A
ASIC	256^3	3.537 s	7.13ms	8.07 HU^2	2.19
Fully Opt.	256^2	2.713 s	5.47ms	8.07 HU^2	1.3
Naive	512^2	42.69 s	86.08 ms	8.04 HU^2	N/A
ASIC	512^3	10.82 s	21.82 ms	8.07 HU^2	3.9
Fully Opt.	512^2	6.076 s	12.25 ms	8.07 HU^2	1.78

Results

Beats the fastest GPU-based FBP implementation

- compare ranking at http://www.rabbitCT.com

RUN TIMES OF THE BEST KNOWN AND FULLY OPTIMIZED CONFIGURATIONS

Configuration	Resolution	Total	Mean	Error	Speed-up
Best Known	256^2	3.843 s	7.75 ms	8.071 HU^2	N/A
Fully Opt.	256^3	2.713 s	5.47 ms	8.071 HU^2	1.4
Best Known	512^3	13.94 s	28.11 ms	8.078 HU^2	N/A
Fully Opt.	512^2	6.076 s	12.25 ms	8.078 HU^2	2.29

See upcoming paper at the *Intern'l High Performance Image Reconstruction Workshop (HPIR)*, July 2011

- E. Papenhausen, Z. Zheng, K. Mueller, "GPU-Accelerated Back-Projection Revisited: Squeezing Performance by Careful Tuning"

Naive

```
#define R_L 2.0f
#define O_L -127.0f

__device__ float f_Ld[128*128*128];

__global__ void gpu_backproject(float *I_nd, float *A_nd){

   int row = blockIdx.y * blockDim.y + threadIdx.y;
   int col = blockIdx.x * blockDim.x + threadIdx.x;

   float u_n;
   float v_n;
   float w_n;

   float result;

   float alpha, beta;
```


Naive

```
for(int k = threadIdx.z*64; k < (threadIdx.z+1)*64; k++){
    w_n = __frcp_rn((A_nd[2] * (O_L + col * R_L) + A_nd[5] * (O_L + row * R_L)+ A_nd[11]) + A_nd[8] * (O_L + k*R_L));
    u_n = (A_nd[0] * (O_L + col * R_L) + A_nd[3] * (O_L + row * R_L) + A_nd[6] * (O_L + (k * R_L)) + A_nd[9]) * w_n;
    v_n = (A_nd[1] * (O_L + col * R_L) + A_nd[4] * (O_L + row * R_L) + A_nd[7] * (O_L + (k * R_L)) + A_nd[10]) * w_n;

    alpha = u_n - (int)u_n;
    beta = v_n - (int)v_n;

    if((int)u_n >=0 && (int)u_n < 1248 && (int)v_n >= 0 && (int)v_n < 960){
        result = (1.0 - alpha) * (1.0 - beta) * I_nd[(int)v_n * 1248 + (int)u_n];
    }
    else{
        result = 0.0f;
    }

    if((int)(u_n+1) >=0 && (int)(u_n+1) < 1248 && (int)v_n >= 0 && (int)v_n < 960){
        result += (alpha) * (1.0 - beta) * I_nd[(int)v_n * 1248 + (int)(u_n+1)];
    }
    else{
        result += 0.0f;
    }
```

Naive

```
if((int)u_n >=0 && (int)u_n < 1248 && (int)(v_n+1) >= 0 && (int)(v_n+1) < 960){
    result += (1.0 - alpha) * (beta) * I_nd[(int)(v_n+1) * 1248 + (int)u_n];
}
else{
    result += 0.0f;
}

if((int)(u_n+1) >=0 && (int)(u_n+1) < 1248 && (int)(v_n+1) >= 0 && (int)(v_n+1) < 960){
    result += (alpha) * (beta) * I_nd[(int)(v_n+1) * 1248 + (int)(u_n+1)];
}
else{
    result += 0.0f;
}

result = ((float)((w_n*w_n) * result));

f_Ld[k * 128 * 128  + row * 128 + col] += result;
```


Optimized

```
texture<float, 2> texRef;

#define R_L 0.5f
#define O_L -127.75f

__constant__ float A_nd[12];

__device__ float f_Ld[512*512*512];

__global__ void gpu_backproject(){

  short row = blockIdx.y * blockDim.y + threadIdx.y;
  short col = blockIdx.x * blockDim.x + threadIdx.x;

  float u_n;
  float v_n;
  float w_n;

  float result;
```


Optimized

```
for(short k = threadIdx.z*256; k < (threadIdx.z+1)*256; k++){
    result = f_Ld[k * 512 * 512 + row * 512 + col];

    w_n = __frcp_rn((A_nd[2] * (O_L + col * R_L) + A_nd[5] * (O_L + row * R_L)+ A_nd[11]) + A_nd[8] * (O_L + k*R_L));
    u_n = (A_nd[0] * (O_L + col * R_L) + A_nd[3] * (O_L + row * R_L) + A_nd[6] * (O_L + (k * R_L)) + A_nd[9]) * w_n;
    v_n = (A_nd[1] * (O_L + col * R_L) + A_nd[4] * (O_L + row * R_L) + A_nd[7] * (O_L + (k * R_L)) + A_nd[10]) * w_n;

    result += ((w_n*w_n) * tex2D(texRef, (u_n + 0.5), (v_n + 0.5)));

    f_Ld[k *512 * 512  + row * 512 + col] = result;
}
```


Best

```
texture<float, 2> texRef;
texture<float, 2> texRef2;
texture<float, 2> texRef3;
texture<float, 2> texRef4;

#define R_L 0.5f
#define O_L -127.75f

__constant__ float A_nd[48];

__device__ float f_Ld[512*512*512];

__global__ void gpu_backproject(){

  short row = blockIdx.y * blockDim.y + threadIdx.y;
  short col = blockIdx.x * blockDim.x + threadIdx.x;

  float u_n;
  float v_n;
  float w_n;

  float result;
```

Best

SPIE Medical Imaging

```
for(short k = threadIdx.z*256; k < (threadIdx.z+1)*256; k++){
    result = f_Ld[k * 512 * 512 + row * 512 + col];

    w_n = __frcp_rn((A_nd[2] * (O_L + col * R_L) + A_nd[5] * (O_L + row * R_L)+ A_nd[11]) + A_nd[8] * (O_L + k*R_L));
    u_n = (A_nd[0] * (O_L + col * R_L) + A_nd[3] * (O_L + row * R_L) + A_nd[6] * (O_L + (k * R_L)) + A_nd[9]) * w_n;
    v_n = (A_nd[1] * (O_L + col * R_L) + A_nd[4] * (O_L + row * R_L) + A_nd[7] * (O_L + (k * R_L)) + A_nd[10]) * w_n;

    result += ((w_n*w_n) * tex2D(texRef, (u_n + 0.5), (v_n + 0.5)));

    w_n = __frcp_rn((A_nd[14] * (O_L + col * R_L) + A_nd[17] * (O_L + row * R_L)+ A_nd[23]) + A_nd[20] * (O_L + k*R_L));
    u_n = (A_nd[12] * (O_L + col * R_L) + A_nd[15] * (O_L + row * R_L) + A_nd[18] * (O_L + (k * R_L)) + A_nd[21]) * w_n;
    v_n = (A_nd[13] * (O_L + col * R_L) + A_nd[16] * (O_L + row * R_L) + A_nd[19] * (O_L + (k * R_L)) + A_nd[22]) * w_n;

    result += ((w_n*w_n) * tex2D(texRef2, (u_n + 0.5), (v_n + 0.5)));

     w_n = __frcp_rn((A_nd[26] * (O_L + col * R_L) + A_nd[29] * (O_L + row * R_L)+ A_nd[35]) + A_nd[32] * (O_L + k*R_L));
     u_n = (A_nd[24] * (O_L + col * R_L) + A_nd[27] * (O_L + row * R_L) + A_nd[30] * (O_L + (k * R_L)) + A_nd[33]) * w_n;
     v_n = (A_nd[25] * (O_L + col * R_L) + A_nd[28] * (O_L + row * R_L) + A_nd[31] * (O_L + (k * R_L)) + A_nd[34]) * w_n;

     result += ((w_n*w_n) * tex2D(texRef3, (u_n + 0.5), (v_n + 0.5)));

     w_n = __frcp_rn((A_nd[38] * (O_L + col * R_L) + A_nd[41] * (O_L + row * R_L)+ A_nd[47]) + A_nd[44] * (O_L + k*R_L));
     u_n = (A_nd[36] * (O_L + col * R_L) + A_nd[39] * (O_L + row * R_L) + A_nd[42] * (O_L + (k * R_L)) + A_nd[45]) * w_n;
     v_n = (A_nd[37] * (O_L + col * R_L) + A_nd[40] * (O_L + row * R_L) + A_nd[43] * (O_L + (k * R_L)) + A_nd[46]) * w_n;

     result += ((w_n*w_n) * tex2D(texRef4, (u_n + 0.5), (v_n + 0.5)));

     f_Ld[k *512 * 512  + row * 512 + col] = result;
 }
```