Nick Black (nickblack@linux.com)
CS4803DGC, Spring 2010, Homework #3
CUDA-optimized convolution

Sizes 16×16, 128×128, 512×512, 512×1024, 1024×1024, 4096×4096, and 8192×8192 were tested using an NVIDIA® GeForce® 8400 GS G98 at 567MHz with 667MHz GDDR2 connected via 32-bit PCI at 33MHz to an Intel® Core™ 2 Duo 6600 at 2.4GHz with 800MHz DDR2. Fixed PCI transfer costs dominated timings[1] and ought be substantially reduced by 16 PCIe lanes.

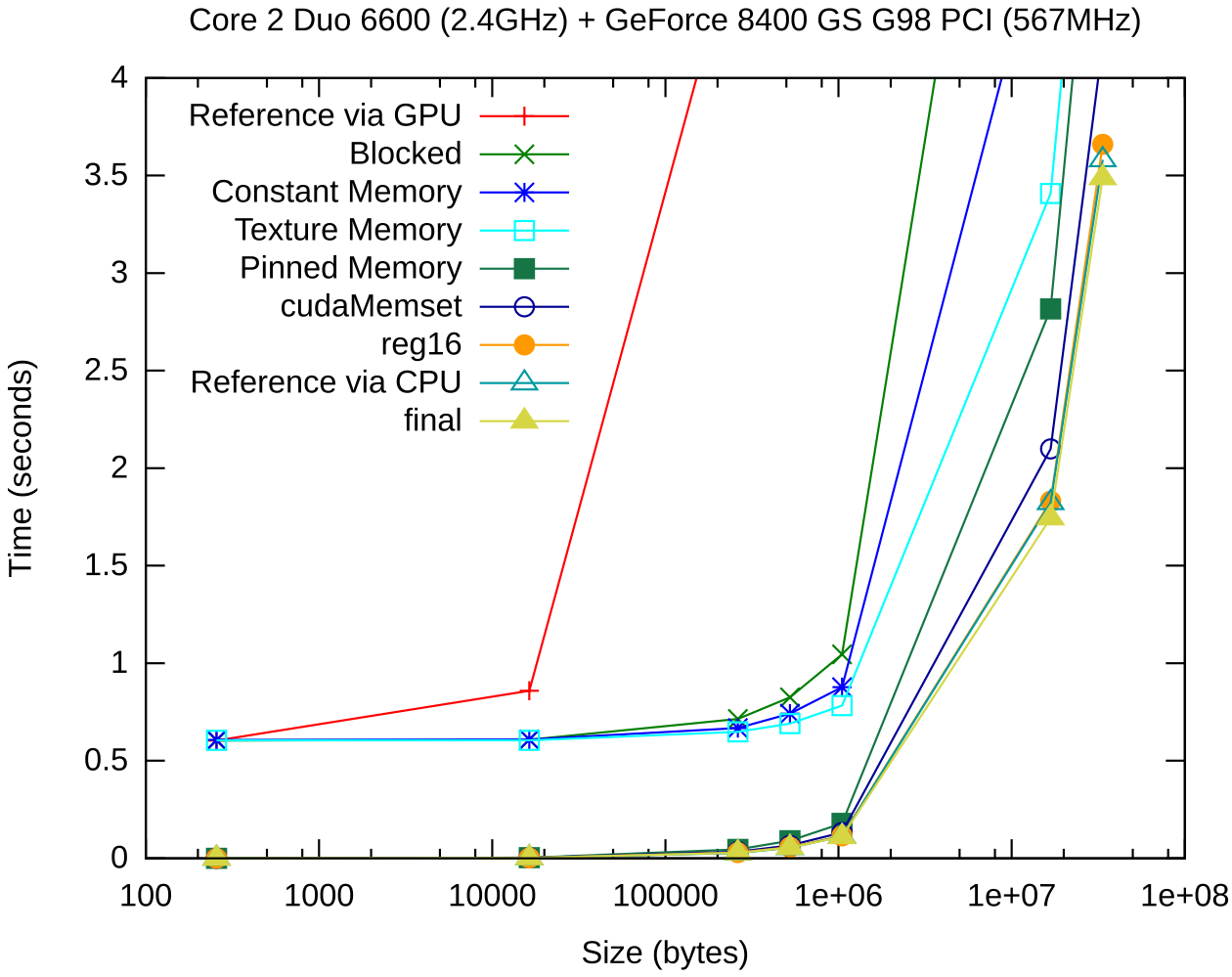

Figure 1: Measurements on `recombinator.qemfd.net`

[1] My final implementation beat out the CPU despite these lengthy delays.

The reference implementation was copied into a kernel body to acquire a CUDA performance baseline, performing significantly worse than the equivalent host code (as expected). The reference implementation was parallelized and blocked at 16×16. These dimensions both conformed to the problem specification[2] and led to coalesced references to image memory. This was the single most significant gain, thanks to massive parallelism.

The 5×5 convolution kernel was moved into on-card constant memory. This provided significant improvement on sufficiently large inputs.

Constant memory declarations

```
__device__ __constant__ float kern[KERNEL_SIZE * KERNEL_SIZE];
```

Constant memory setup

```
CUDA_SAFE_CALL(cudaMemcpyToSymbol("kern", M.elements,
                        sizeof(*M.elements) * M.height * M.width,
                        0, cudaMemcpyHostToDevice));
```

The input image was moved into on-card texture memory. This further improved upon the constant memory solution: kernel time represented less than 10% of absolute costs.

Texture memory declarations

```
cudaArray *texmat = NULL;
cudaChannelFormatDesc texdesc;
texture<float, 2, cudaReadModeElementType> texref;
```

Texture memory setup

```
texdesc = cudaCreateChannelDesc(32,0,0,0,cudaChannelFormatKindFloat);
CUDA_SAFE_CALL(cudaMallocArray(&texmat,&texdesc,N.width,N.height));
CUDA_SAFE_CALL(cudaMemcpyToArray(texmat,0,0,N.elements,
                N.width * N.height * sizeof(*N.elements),
                cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaBindTextureToArray(texref,texmat));
```

Kernel employing texture memory

```
sum += kern[m * KERNEL_SIZE + n] *
                tex2D(texref,(float)(x + n - 2),(float)(y + m - 2));
```

Host memory involved in bus transfers was pinned with write-combining. This eliminated an internal bounce buffer (at the cost of overall system resources), dramatically cutting absolute time[3].

Pinning allocation replacements

```
CUDA_SAFE_CALL(cudaHostAlloc(&M.elements,size * sizeof(*M.elements),0)); // in AllocateMatrix()
CUDA_SAFE_CALL(cudaFreeHost(M->elements));                               // in FreeMatrix()
```

Replacing the zero-via-copy operation with `cudaMemset()` eliminated a further 20% of runtime.

cudaMemset

```
CUDA_SAFE_CALL(cudaMemset(Pd.elements,0,
        Pd.height * Pd.width * sizeof(*Pd.elements)));
```

`cudaprof` reported occupancy of only 33%[4]. By slightly restructuring the kernel, registers per thread were cut from 17 to 16 (ultimately 14), boosting occupancy to 67%. Duff's Device[5] resulted in a ˜30% reduction in dynamic instructions and another ˜8% net savings.

[2] All sizes were guaranteed to be multiples of 16.

[3] `--ptx` was used to generate PTX assembly for analysis.

[4] See Figures 2 and 3 for `cudaprof` output.

[5] See this famous comp.lang.c post.

Device information for recombinator

```
[recombinator](0) $ ~/local/cuda/C/bin/linux/release/deviceQuery
CUDA Device Query (Runtime API) version (CUDART static linking)
There is 1 device supporting CUDA

Device 0: "GeForce 8400 GS"
  CUDA Driver Version:                           2.30
  CUDA Runtime Version:                          2.30
  CUDA Capability Major revision number:         1
  CUDA Capability Minor revision number:         1
  Total amount of global memory:                 536608768 bytes
  Number of multiprocessors:                     1
  Number of cores:                               8
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          262144 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1.40 GHz
  Concurrent copy and execution:                 No
  Run time limit on kernels:                     No
  Integrated:                                    No
  Support host page-locked memory mapping:       No
  Compute mode:                                  Default
        (multiple host threads can use this device simultaneously)

Test PASSED

Press ENTER to exit...

[recombinator](0) $
```

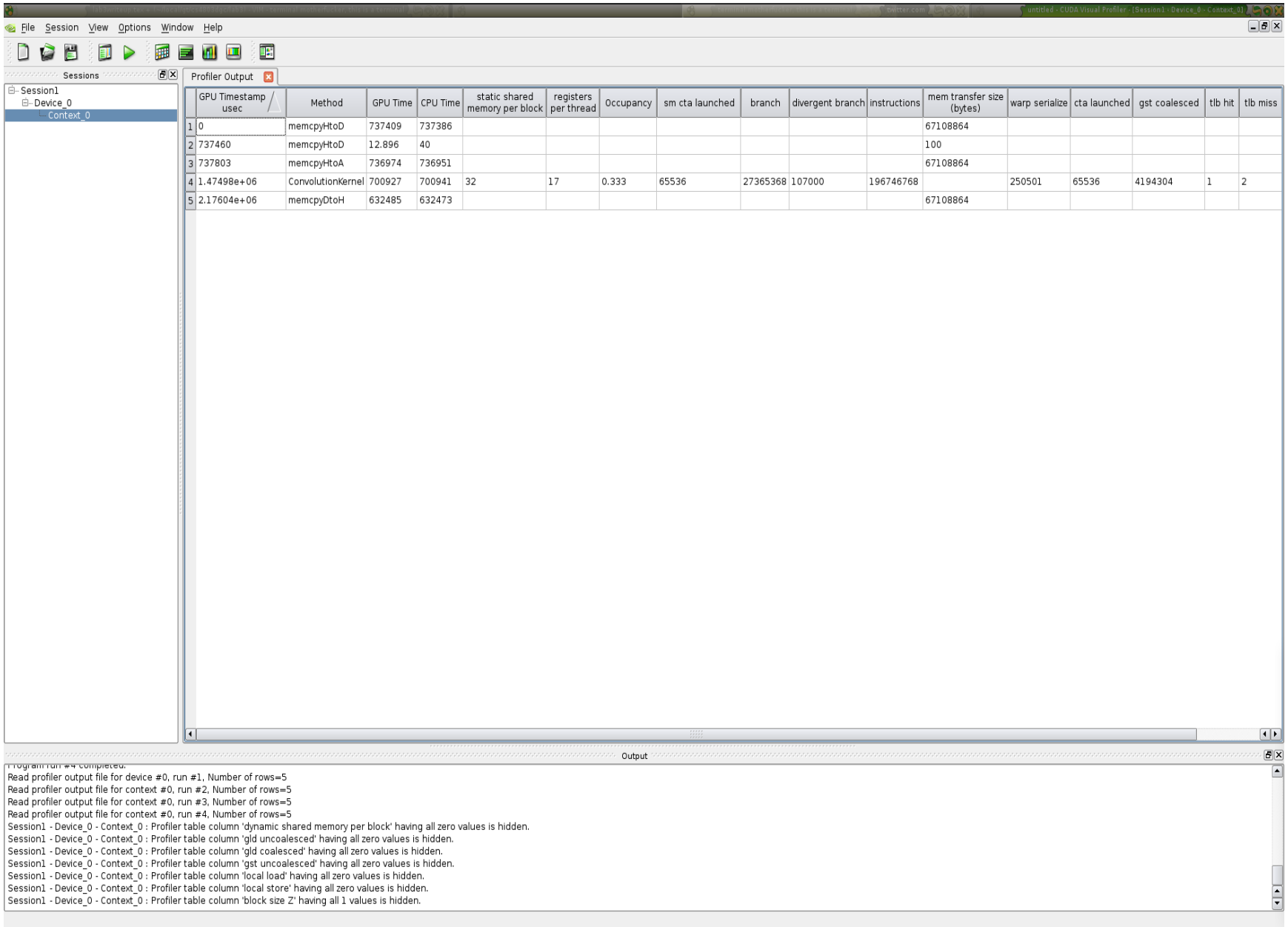

Figure 2: `cudaprof` output following memory pinning optimization

Figure 3: `cudaprof` output following occupancy-boosting optimization