GPU computing
CUDA

Rossen Radev
29-01-2008
CPU vs GPU

CPU

GPU

- Control
- ALU
- ALU
- Cache
- DRAM

- Shared
- DRAM
NVIDIA-CUDA

- CUDA enabled GPU products
  - GeForce 8400-8800
  - Quadro
  - Tesla

- G80
  - (1x8) - (4x16x8) processors
  - 128MB-4x1.5GB DRAM
  - Memory Bandwidth 9.6-110 (GB/sec)
CUDA

- CUDA™ Toolkit for C
  - nvcc C compiler
  - Profiler
  - gdb debugger (GPU)?
  - CUDA runtime driver
  - FFT and BLAS libraries
  - OpenGL and DirectX

- Linux 32/64-bit
- Windows XP 32/64-bit

- Other
  - MathWorks MATLAB® Plug-in
  - FORTRAN
Memory Model

CPU is host, GPU is device

host memory

HOST

device memory

device
C extensions

Function type qualifiers

__host__

HOST

__global__

DEVICE

__device__
C extensions

Variable Type Qualifiers
• __device__  (global memory)
• __constant__
• __shared__

__global__ void testKernel( int* gi, int* go)
{
    extern __shared__ int sdata[];
    ...
}

CUDA

Built-in Variables
- `gridDim`
- `blockIdx`
- `blockDim`
- `threadIdx`

Synchronization
- `__syncthreads()`

Mathematical functions
- `Standard C/C++ math library`

Arithmetic Instructions
IEEE-754 deviations

- FMAD
- Division is implemented via the reciprocal.
- Square root is implemented via the reciprocal square root in a non-standard-compliant way.
- For addition and multiplication, only round-to-nearest-even and round-towards-zero are supported via static rounding modes; directed rounding towards +/- infinity is not supported.
IEEE-754 deviations

- Denormalized numbers are not supported; floating-point arithmetic and comparison instructions convert denormalized operands to zero prior to the floating-point operation
- Underflowed results are flushed to zero
- There is no mechanism for detecting that a floating-point exception has occurred and floating-point exceptions are always masked, but when an exception occurs the masked response is standard compliant.
- Signaling NaNs are not supported
MC example

- RNG
  - long period
  - data parallel
  - independent random streams
  - small memory requirements
  - MT parallel – $2^{607}$ period, 4096 threads
MC example

**π** calculation

c1 – in circle

c2 – outside circle
int main()
{
    int
    c1=0,
    c2=0;
    srandom(123);

    for(int i=1; i<=mcsteps; i++)
    {
        double x=rnd();
        double y=rnd();
        if(x*x+y*y < 1.0) c1++;
        else         c2++;
    }

    res = 4.0*(double)c1/(double)(c1+c2);

    printf("%f\n", res);

    exit(0);
}
int main(int argc, const char **argv){
    CUT_DEVICE_INIT();
    CUDA_SAFE_CALL( cudaMalloc((void **)&d_Random, RAND_N * sizeof(float)) );

    RandomGPU<<<32, 128>>>(d_Random, N_PER_RNG, 123);
    CUDA_SAFE_CALL( cudaThreadSynchronize() );

    CUDA_SAFE_CALL( cudaMalloc( (void**) &d_odata, mem_size) );
    PiGPU<<<32, 128>>>(d_Random, d_odata, MT_RNG_COUNT, N_PER_RNG);
    CUDA_SAFE_CALL( cudaThreadSynchronize() );
    CUDA_SAFE_CALL( cudaMemcpy( h_odata, d_odata, sizeof(int) * num_threads, cudaMemcpyDeviceToHost) );

    int c1 = 0;
    for(int i=0; i< num_threads; i++){
        c1+=h_odata[i];
    }
    res = 8.0*(double)c1/(double)(RAND_N);

    CUDA_SAFE_CALL( cudaFree(d_Random) );
    CUDA_SAFE_CALL( cudaFree(d_odata) );
    CUT_SAFE_CALL( cutDeleteTimer( hTimer) );

    CUT_EXIT(argc, argv);
}
__global__ void PiGPU(float *d_Random, int* d_odata, int rng_count, int nperrng) {

    const int tid = blockDim.x * blockIdx.x + threadIdx.x;
    const int thread_n = blockDim.x * gridDim.x;

    int c1=0;
    float xx, yy;
    for(int iRng = tid; iRng < rng_count; iRng += thread_n) {
        for(int iOut = 0; iOut < nperrng; iOut+=2) {
            xx = d_Random[iRng + (iOut + 0) * rng_count];
            yy = d_Random[iRng + (iOut + 1) * rng_count];
            if(xx*xx + yy*yy < 1.0f)
                c1++;
        }
    }

    // write data to global memory
    d_odata[tid] = c1;
}

Applications

- Molecular Dynamics
- Monte Carlo
- Signal Processing
- Computational biology string matching
- Interactive Visualization with Volume Rendering
- Numerical Weather Prediction
- Folding@home
Applications

- Finance (MC)
- Geographic Information Systems (GIS)
- Electromagnetic Simulations & Geophysical Data Processing
- 3D visualization solutions for Oil and Gas
- Synthesis of Artificial Neural Circuitry
FluidsGL
GPU Acceleration of Molecular Modeling Applications

University of Illinois - John Stone  http://www.ks.uiuc.edu/Research/gpu/

NAMD

VMD
That's all folks

Questions?