L10: Dense Linear Algebra on GPUs

Administrative Issues

- Next assignment, triangular solve
  - Due 5PM, Friday, March 5
  - handin cs6963 lab 3 <probfile>"
- Project proposals (discussed today)
  - Due 5PM, Wednesday, March 17 (hard deadline)
  - A few minutes at end of class to help form groups

Outline

- Triangular solve assignment
- Project
  - Ideas on how to approach
  - Construct list of questions

Reading:

Paper: Volkov, V., and Demmel, J. W. 2008. Benchmarking GPUs to tune dense linear algebra, *SC08*, November 2008.

Paper link: http://portal.acm.org/citation.cfm?id=1413402

Talk link: http://www.eecs.berkeley.edu/~volkov/volkov08-sc08talk.pdf

Volkov code: http://forums.nvidia.com/index.php?showtopic=47689&st=40&p=314014&#entry314014

Triangular Solve (STRSM)

```
for (j = 0; j < n; j++)
   for (k = 0; k < n; k++)
       if (B[j*n+k] != 0.0f) {
            for (i = k+1; i < n; i++)
              B[j*n+i] -= A[k * n + i] * B[j * n + k];
       }
```

Equivalent to:

```
cublasStrsm('l' /* left operator */, 'l' /* lower triangular */,
            'N' /* not transposed */, 'u' /* unit triangular */,
             N, N, alpha, d_A, N, d_B, N);
```

See: http://www.netlib.org/blas/strsm.f

Assignment

- Details:
  - Integrated with simpleCUBLAS test in SDK
  - Reference sequential version provided

1. Rewrite in CUDA
2. Compare performance with CUBLAS 2.0 library

Performance Issues?

- + Abundant data reuse
- - Difficult edge cases
- - Different amounts of work for different <j,k> values
- - Complex mapping or load imbalance

Reminder: Outcomes from Last Year's Course

- Paper and poster at Symposium on Application Accelerators for High-Performance Computing http://saahpc.ncsa.illinois.edu/09/ (May 4, 2010 submission deadline)
  - Poster:
    Assembling Large Mosaics of Electron Microscope Images using GPU - Kannan Venkataraju, Mark Kim, Dan Gerszewski, James R. Anderson, and Mary Hall
  - Paper:
    GPU Acceleration of the Generalized Interpolation Material Point Method
    Wei-Fan Chiang, Michael DeLisi, Todd Hummel, Tyler Prete, Kevin Tew, Mary Hall, Phil Wallstedt, and James Guilkey
- Poster at NVIDIA Research Summit
  http://www.nvidia.com/object/gpu_tech_conf_research_summit.html
  Poster #47 - Fu, Zhisong, *University of Utah (United States)*
  Solving Eikonal Equations on Triangulated Surface Mesh with CUDA
- Posters at Industrial Advisory Board meeting
- Integrated into Masters theses and PhD dissertations
- Jobs and internships

Projects

- 2-3 person teams
- Select project, or I will guide you
  - From your research
  - From previous classes
  - Suggested ideas from faculty, Nvidia (ask me)
- Example (published):
  - http://saahpc.ncsa.illinois.edu/09/papers/Chiang_paper.pdf (see prev slide)
- Steps
  1. Proposal (due Wednesday, March 17)
  2. Design Review (in class, April 5 and 7)
  3. Poster Presentation (last week of classes)
  4. Final Report (due before finals)

1. Project Proposal (due 3/17)

- Proposal Logistics:
  - Significant implementation, worth 55% of grade
  - Each person turns in the proposal (should be same as other team members)
- Proposal:
  - 3-4 page document (11pt, single-spaced)
  - Submit with handin program:
    "handin cs6963 prop <pdf-file>"

Content of Proposal

I. Team members: Name and a sentence on expertise for each member

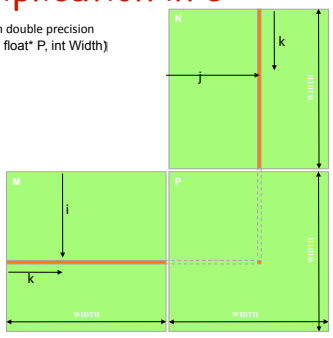
II. Problem description
- What is the computation and why is it important?
- Abstraction of computation: equations, graphic or pseudo-code, no more than 1 page

III. Suitability for GPU acceleration
- Amdahl's Law: describe the inherent parallelism. Argue that it is close to 100% of computation. Use measurements from CPU execution of computation if possible.
- Synchronization and Communication: Discuss what data structures may need to be protected by synchronization, or communication through host.
- Copy Overhead: Discuss the data footprint and anticipated cost of copying to/from host memory.

IV. Intellectual Challenges
- Generally, what makes this computation worthy of a project?
- Point to any difficulties you anticipate at present in achieving high speedup

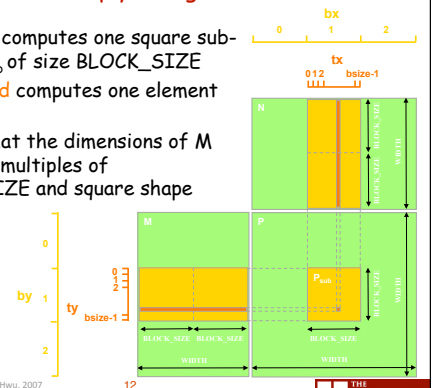
Reminder from L5: Matrix Multiplication in C

```
// Matrix multiplication on the (CPU) host in double precision
void MatrixMulOnHost(float* M, float* N, float* P, int Width)
{
  for (int i = 0; i < Width; ++i)
    for (int j = 0; j < Width; ++j) {
      double sum = 0;
      for (int k = 0; k < Width; ++k) {
        double a = M[i * width + k];
        double b = N[k * width + j];
        sum += a * b;
      }
      P[i * Width + j] = sum;
    }
}
```



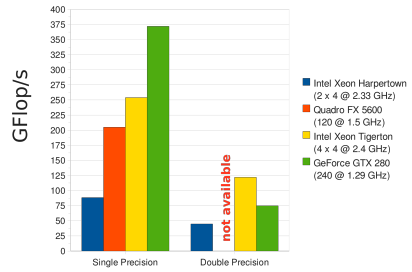
Tiled Matrix Multiply Using Thread Blocks

- One block computes one square sub-matrix P_{sub} of size BLOCK_SIZE
- One thread computes one element of P_{sub}
- Assume that the dimensions of M and N are multiples of BLOCK_SIZE and square shape



GEMM on Current CPUs and GPUs

(Intel multicores and NVIDIA GPUs)



Note that in SP the GTX 280 is **10 x faster** than a quad-core processor (at 2.33 GHz) and still **75 GFlop/s** faster than an entire quad-socket quad-core Intel Xeon Tigerton system (cores running at 2.4 GHz)

Slide source: Stan Tomov, UTK (see *www.cs.utk.edu/~dongarra)*

Preview: Dense Linear Algebra on GPUs

- SGEMM result is not from algorithm of L5
- Why? Significant reuse can be managed within registers
- Comparison:

	GPU Rule-of-Thumb	This Lecture (SGEMM)
Threading	Generate lots of threads (up to 512/block) to hide memory latency	Only 64 threads/block provides 2 warps, sufficient to hide latency plus conserves registers
Shared memory	Use to exploit reuse across threads	Use to communicate shared data across threads
Registers	Use for temporary per-thread data	Use to exploit significant reuse within a thread

Volkov Slides 3-17, 24

My Research (with USC/ISI and Argonne): What is CHiLL?

High-level loop transformation and code generation framework
- based on polyhedral model
- script interface for programmers or compilers
- optimization strategy expressed as sequence of composable transformations

```
for(i=0; i<10; i++)┐   mxm.c (also supports Fortran as input)
  for(j=0; j<10; j++)┐
    for(k=0; k<10; k++)┐
          c[i][j]+=a[i][k]*b[k][j];
```

CHiLL

213-122.script

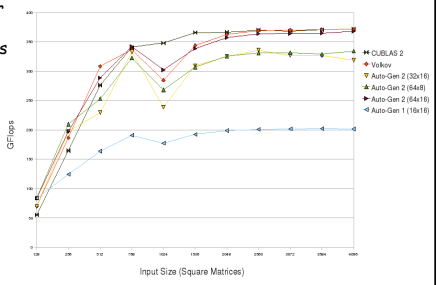
source: mxm.c
procedure: 0
loop: 0

permute([2,1,3])
unroll(0,2,2)
unroll(0,3,2)

```
for(j=0; j<10; j++)┐   mxm213.out
  for(i=0; i<10; i+=2)┐
    for(k=0; k<10; k+=2){
          c[i][j]+=a[i][k]*b[k][j];
          c[i][j]+=a[i][k+1]*b[k+1][j];
          c[i+1][j]+=a[i+1][k]*b[k][j];
          c[i+1][j]+=a[i+1][k+1]*b[k+1][j];
  }
```

Latest Research: CUDA-CHiLL

- Automatically generate CUDA for NVIDIA GPU from sequential code plus script
- Abstraction permits parallel threads & staging of data
- Heterogeneous code generation: Alternative scripts generate CUDA, OpenMP or sequential code tuned for memory hierarchy



Results provided by Malik Khan, Gabe Rudy, Chun Chen

CUDA-CHiLL Automatically-Generated Code

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      c[j][i] += a[k][i]*b[j][k];
```

```
original()
2 tile(0,1,TI,1,counted)
3 tile(0,3,TJ,2,counted)
4 tile(0,5,TK,3,strided)
5 tile(0,4,TK,4,counted)
6 tile(0,5,1,5,counted)
7 tile(0,5,1,4,counted)
8 datacopy privatized(0,3,c,[4,5],false,-1,1,1)
9 datacopy(0,4,b,false,0,1,-16)
10 tile(3,4,(TJ*TK)/TI,4,counted)
11 tile(3,6,1,4,counted)
12 unroll(1,5,0)
13 unroll(2,5,0)
14 unroll(3,6,0)
15 unroll(0,8,0)
16 unroll(0,9,0)
17 cudaize(mm GPU,0,1,n/TI,n/TJ,n,global)
18 cudathread([0,1],TJ,TI/TJ)
19 cudathread([1,1,0,1],TJ,TI/TJ,sync)
20 cudathread([1,1,1,1],TJ,TI/TJ,sync)
21 cudathread([2,1],TJ,TI/TJ)
```

←script

```
float P1[16];
__shared__ float P2[16][17];
bx = blockIdx.x, by = blockIdx.y;
tx = threadIdx.x, ty = threadIdx.y;
P1[0:15] = c[16*by:16*by+15][tx+64*bx+16*ty];
for (t6 = 0; t10 <= 1008; t6+=16) {
  P2[tx][4*ty:4*ty+3] = b[16*by+4*ty:16*by+4*ty+3]
[tx+t6];
  __syncthreads();
  P1[0:15] += a[t6][64*bx+16*ty+tx]*P2[0][0:15]
  P1[0:15] += a[t6+1][64*bx+16*ty+tx]*P2[1][0:15]
  ...
  P1[0:15] += a[t6+15][64*bx+16*ty+tx]*P2[15][0:15]
  __syncthreads();
}
c[16*by:16*by+15][tx+64*bx+16*ty] = P1[0:15];
```

↑Automatically-generated CUDA code

Results provided by Malik Khan, Gabe Rudy, Chun Chen

CUDA-CHiLL: Higher-Level Interface (in progress)

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      c[j][i] += a[k][i]*b[j][k];
```

```
init("mm.sp2", "MarkedLoop")

tile_control({"i","j"}, {TI,TJ},
             {l1_control="ii", l2_control="jj"},
             {"ii", "jj", "i", "j"})
tile_control({"k"}, {TK}, {l1_control="kk"},
             {"ii", "jj", "kk","i", "j","k"}, strided)
tile_control({"i"}, {TJ},
             {l1_control="ty",l1_tile="tx"},
             {"ii", "jj", "kk","tx","ty","j","k"})

--Assign loop levels to thread space and name the kernel
cudaize("mm_GPU",
        {a=N*N, b=N*N, c=N*N},--array sizes for data copying
        {block={"ii","jj"}, thread={"tx","ty"}})
--Copy the "c" array usage to registers
copy_to_registers("kk", "c", {"tx","ty"})
copy_to_shared("ty", "b")
--Unroll two innermost loop levels fully
unroll_to_depth(2)
```

Gabe Rudy Master's thesis

Final Result: Copy C to registers, B to shared memory and unroll

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      c[j][i] += a[k][i]*b[j][k];
```

```
Steps 1 through 4 tile (for computation, data)
--Copy the "c" array usage to registers
5. copy_to_registers("kk", "c", {"tx","ty"})
6. copy_to_shared("ty", "b")
--Unroll two innermost loop levels fully
7. unroll_to_depth(2)
```

B goes into shared memory

C goes into registers and is copied back at end

```
float P1[16];
__shared__ float P2[16][17];
bx = blockIdx.x, by = blockIdx.y;
tx = threadIdx.x, ty = threadIdx.y;
P1[0:15] = c[16*by:16*by+15][tx+64*bx+16*ty];
for (t6 = 0; t10 <= 1008; t6+=16) {
  P2[tx][4*ty:4*ty+3] = b[16*by+4*ty:16*by+4*ty+3]
[tx+t6];
  __syncthreads();
  P1[0:15] += a[t6][64*bx+16*ty+tx]*P2[0][0:15]
  P1[0:15] += a[t6+1][64*bx+16*ty+tx]*P2[1][0:15]
  ...
  P1[0:15] += a[t6+15][64*bx+16*ty+tx]*P2[15][0:15]
  __syncthreads();
}
c[16*by:16*by+15][tx+64*bx+16*ty] = P1[0:15];
```

Next Class

- See Khan/Rudy poster on Friday!
- More complex dense linear algebra
- Some specific global synchronization strategies to support these