Out-of-core algorithms for dense matrix factorization on GPGPU

Ed D'Azevedo
Oak Ridge National Laboratory
dazevedoef@ornl.gov

Amy Huang
Chinese University of Hong Kong
s0961126@cuhk.edu.hk

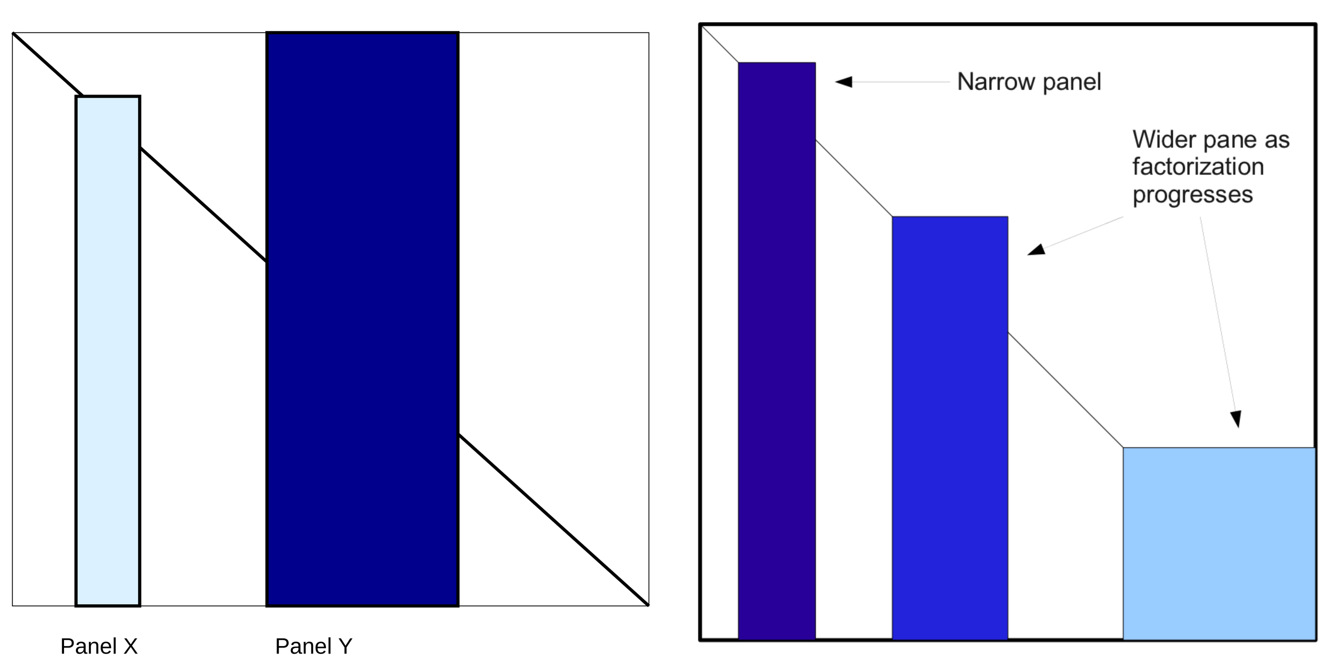
Kwai Wong
University of Tennessee
kwong@utk.edu

Watson Wu
Chinese University of Hong Kong
watsonwu@cuhk.edu.hk

Background

- Solution of large dense matrix problems arises from diverse applications such as modelling the response of heating of fusion plasma to radio frequency (RF) wave, modelling radiation heat transfer, boundary element method, and large scale least squares problem.
- General Purpose Graphics Processing Unit (GPGPU) offers performance several times faster than multi-core CPU. GPGPU has dedicated memory to provide very high memory bandwidth. Even inexpensive (less than $200) consumer-grade video cards contain Nvidia Fermi GPGPU processor with 1 GBytes of device memory. However, the PCI connection can be bottleneck in transferring data between GPGPU device and CPU host.
- The MAGMA Library [2] achieves very high performance on GPGPU for dense matrix computations. However, the largest problem size is limited to the amount of local device memory on GPGPU.
- **Idea:** Adapt out-of-core algorithms for solving large problems on GPGPU so that a matrix of size say 10 GBytes can still be factored on GPGPU with only 1 GBytes of device memory.

Cholesky factorization



(a) Factorization algorithm uses 2 column panels. (b) Panel width increases for the same amount of device memory.

- Organise matrix as wide column panels that still fit on GPGPU (see Figure).
- Use left-looking out-of-core algorithm to minimise the amount of data transfer [1]. Cholesky factorization access only lower triangular part
- Block algorithm

$$\begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L'_{11} & L'_{21} \\ & L'_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A'_{21} \\ A_{21} & A_{22} \end{pmatrix}$$

1. Cholesky factorization of diagonal block (`magma_dpotrf_gpu`)

$$L_{11}L'_{11} = A_{11}$$

2. Update column panel (`cublasDtrsm`)

$$L_{21} = A_{21}\,L'_{11}$$

3. Symmetric update of unfactored submatrix (`cublasDsyrk`)

$$\tilde{A}_{22} \leftarrow A_{22} - L_{21}L'_{21}$$

4. Cholesky factorization of updated submatrix

$$L_{22}L'_{22} = \tilde{A}_{22}$$

- Use right-looking algorithm MAGMA `magma_dpotrf` for factorization of diagonal block of panel Y in device memory of GPGPU
- As the factorization proceeds, widths of panels X and Y are increased to use available device memory.

LU factorization

- Similar to Cholesky factorization but access to full matrix using fixed size panels
- If panel Y has width (N/K), then need extra $O(KN^2)$ transfers. Therefore panel Y takes up most available memory to be as wide as possible.
- MAGMA `magma_dgetrf` designed for nearly square system and need $\max(M,N)^2$ amount of GPGPU memory.
- Hybrid factorization algorithm: LU factorization of narrow column panel performed on multi-core CPU host using LAPACK `DGETRF`, right-looking update performed by GPGPU using CUBLAS.

Results

The out-of-core algorithms were tested using only 1 GB (out of 5 GB) ***on purpose*** to highlight the message that substantial performance is achieved *even* with only a small amount of device memory.

	MKL (12 CPU)	MAGMA 1.0	Out-of-core algorithm
N=25,000	121 Gflops/s	271 Gflops/s	200 Gflops/s
N=35,000	123 Gflops/s	Out of memory	214 Gflops/s

Table 2: Comparison of MAGMA 1.0 to out-of-core LU factorization (DGETRF) using only 1 GBytes out of 5 GBytes of Nvidia M2070.

	MKL (12 CPU)	MAGMA 1.0	Out-of-core algorithm
N=25,000	122 Gflops/s	266 Gflops/s	246 Gflops/s
N=35,000	123 Gflops/s	Out of memory	263 Gflops/s

Table 3: Comparison of MAGMA 1.0 to out-of-core Cholesky factorization (DPOTRF) using only 1 GBytes out of 5 GBytes of Nvidia M2070.

Environment

- Numerical experiments performed on one compute node on Keeneland [3].
- Each compute node is an HP ProLiant SL390s G7 with two Intel Xeon X5660 Westmere 6 core processor(total 12 cores) and 24 GBytes of memory.
- Each node has three Nvidia Tesla M2070 (Fermi) GPGPUs, but only one GPGPU was used. Each GPGPU has 5 GBytes of device memory and full PCIe X16 bandwidth.
- Nvidia M2070: double precision (peak) 515 Gflops/sec, single precision (peak) 1.03 Tflops/sec, memory bandwidth 148 GB/sec , 1147.0 MHz, 5375.4 MB memory
- Software libraries: CUDA 4.0, CUBLAS, MAGMA 1.0 (built for Fermi), Intel MKL 10.0.1.014.
- DGEMM matrix multiply (N=9533): MAGMA 303 Gflops/sec, CUBLAS 296 Gflops/sec. Effective transfer rate approximately 6 GB/sec each way (using `cublasSetMatrix()` and `cublasGetMatrix()` on $10,000 \times 10,000$ double precision matrix)

```
for( ystart=1; ystart <= n; ystart = yend + 1) {
   // determine nby = width for panel Y
   yend = min(n, ystart+nby-1); ysize = yend - ystart + 1;

   // Transfer from CPU to GPGPU
   Y(ystart:n,1:ysize)  = A(ystart:n,ystart:yend);

   for( xstart=1; xstart <= (ystart-1); xstart = xend + 1) {
     // determine nbx = width for panel X
     xend = min((ystart-1),xstart+nbx-1);
     xsize = xend - start + 1;

     // Transfer from CPU to GPGPU
     X(ystart:n,1:xsize) = A(ystart:n,xstart:xend);

     // perform part of symmetric update (DSYRK)
     // A22 <- A22 - L21*L21'

     Y(ystart:yend,1:ysize) = Y( ystart:yend,1:ysize) -
      X(ystart:yend,1:xsize)*transpose(X(ystart:yend,1:xsize));

    // update lower part of Y (DGEMM)
    Y((yend+1):n,1:ysize) = Y( (yend+1):n,1:ysize) -
     X( (yend+1):n,1:xsize)*transpose(X( ystart:yend,1:xsize));
    };

   // all previous updates complete
   // ready to process panel Y

   // factor diagonal block (DPOTRF)
   Y(start:yend,1:ysize) = chol( Y(ystart:yend,1:ysize) )

   // update lower part using triangular solve (DTRSM)

   Y((yend+1):n,1:ysize) =
      Y((yend+1):n,1:ysize)/transpose(Y(ystart:yend,1:ysize));

   // copy Y panel from GPGPU to CPU host
   A(ystart:n,ystart:yend) = Y( ystart:n,1:ysize );
};
```

Figure 2: High-level description of out-of-core Cholesky factorization.

Acknowledgements

The submitted manuscript has been authored by a contractor of the U.S. Government under Contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. The NSF Keeneland Project (National Science Foundation award OCI-0910735) provided computing resources for this project. Amy Huang and Watson Wu were supported by a fellowship from the Department of Mathematics, the Chinese University of Hong Kong.

References

[1] E. F. D'Azevedo and J. Dongarra, *The design and implementation of the parallel out-of-core ScaLAPACK LU, QR, and Cholesky factorization routines*, Concurrency: Practice and Experience, 12 (2000), pp. 1481–1493.

[2] S. Tomov, *MAGMA – LAPACK for GPUs*, Presented at Keenland GPU Tutorial 2011, Atlanta, GA, April 14-15, 2011. http://icl.cs.utk.edu/magma/.

[3] J. S. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford, and S. Yalamanchili, *Keeneland: Bringing heterogeneous GPU computing to the computational science community*, IEEE Computing in Science and Engineering, 13 (2011), pp. 90–5. http://keeneland.gatech.edu/.