GPU ACCELERATION OF WSMP
(WATSON SPARSE MATRIX PACKAGE)

NATALIA GIMELSHEIN  NVIDIA
ANSHUL GUPTA       IBM
STEVE RENNICH      NVIDIA
SEID KORIC         NCSA
WATSON SPARSE MATRIX PACKAGE (WSMP)

- Cholesky, LDLᵀ, LU factorization
- Distributed- and shared-memory algorithms
- Scalable, employs theoretically most scalable factorization algorithm
- Uses multifrontal method
- This work concentrates on accelerating numerical factorization phase
OBJECTIVE

- Acceleration of WSMP
  - Demonstrate suitability of GPUs for Sparse Direct Solves
  - Demonstrate suitability of GPUs for very irregular workloads
- Evaluate methods for accelerating WSMP
  - Simple methods can work well
  - More sophisticated methods can work better
OUTLINE

- Minimally invasive acceleration of BLAS-intensive applications
- High level interface
- Acceleration techniques
- MPI acceleration
GPU ACCELERATION: MINIMALLY INVASIVE APPROACH

- Intercepting BLAS level 3 calls with big dimensions and sending them to GPU
- Tiling to hide copies
- PCIe and host memory bandwidth are limiting
- But many of the matrices are not big enough to be tiled (sizes)
GPU ACCELERATION: MINIMALLY INVASIVE APPROACH

- We are intercepting simultaneous moderate-size calls
  - each one is not big enough to fully occupy GPU
- Use streams to
  - Increase GPU utilization
  - Hide copy-up and copy-down
  - Kernels may overlap
GPU ACCELERATION: MINIMALLY INVASIVE APPROACH

- Use host pinned buffers to increase copy-up,copy-down speed and enable asynchronous memory copies
- Send small BLAS calls to the CPU
  - $(m,n<128, k<512)$
- Large matrices are tiled individually
- Can be used with ANY application, not just WSMP
- No recompilation of the code required
RESULTS - SYSTEM USED

- Dual-socket Ivy-Bridge Xeon @ 3.0 Ghz
  - 20 cores total, PCIe gen3, E5-2690 v2
- Tesla K40, ECC on
- Intel MKL for host BLAS calls
- CUDA 6.5
- Standard cuBLAS routines are used for BLAS and BLAS-Like.
RESULTS - DROP-IN GPU ACCELERATION

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Performance, Gflops/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>audkw_1</td>
<td>1.48</td>
</tr>
<tr>
<td>bone010</td>
<td>1.45</td>
</tr>
<tr>
<td>Emilia_923</td>
<td>1.66</td>
</tr>
<tr>
<td>Fault_639</td>
<td>1.53</td>
</tr>
<tr>
<td>Flan_1565</td>
<td>1.37</td>
</tr>
<tr>
<td>Geo_1438</td>
<td>1.67</td>
</tr>
<tr>
<td>Hook_1498</td>
<td>1.77</td>
</tr>
<tr>
<td>ldoor</td>
<td>0.88</td>
</tr>
<tr>
<td>nd24k</td>
<td>1.30</td>
</tr>
<tr>
<td>Serena</td>
<td>1.82</td>
</tr>
</tbody>
</table>

2 x Xeon E5-2690 v2 + K40 (max boost, ECC=on)
SPARSE DIRECT SOLVERS

- **Supernodes**
  - collections of columns with similar non-zero pattern
  - provide opportunity for dense matrix math
  - grow with mesh size due to ‘fill’
  - The larger the model, the larger the supernodes
  - Supernodes in the factors are detected during symbolic factorization
DENSE BLOCK CHOLESKY

- Basis for sparse direct algorithms
- Emphasizes dense math
- Dominated by computation of Schur complement

\[ A_{11} \quad A_{21}^t \]
\[ A_{21} \quad A_{22} \]

\[ L_{11} \quad 0 \]
\[ 0 \quad I \]

\[ L_{21} \quad I \]

\[ L_{11}^t \quad L_{11}^{t} \]
\[ L_{21} \]

\[ X \]

\[ L_{21} \quad 0 \]

\[ 0 \quad I \]

\[ A_{22}^* = A_{22} - L_{21} L_{21}^t \]

- POTRF - element-wise Cholesky factorization
- TRSM - triangular solve
- SYRK

\[ L_{11} L_{11}^t = A_{11} \]
\[ L_{21} L_{11}^t = A_{21} \]
\[ A_{22}^* = A_{22} - L_{21} L_{21}^t \]
PARALLEL MULTIFRONTAL METHOD

- A task that owns the supernode
  - Assembles frontal matrix
  - Factors the supernode
  - Computes update frontal matrix, that will be used by the parent task
- In the beginning, many independent supernodes to be factored by parallel threads
- In the end, threads cooperate on the factorization of fewer remaining supernodes
Potential for Improvement

- Potentially big BLAS calls are split into smaller ones, hurting performance
  - Same data is moved back and forth many times
  - A lot of pressure on PCIe for data movement
  - A lot of pressure on host memory bandwidth
- A few GB of memory allocated on the device and has to be pinned on the host
SOLUTION - HIGH LEVEL INTERFACE

- Most of the work in Cholesky factorization is performed in dtrsmesyrk calls (dtrsm followed by dsyrk)
- Bigger BLAS dimensions are more favorable for the GPU
- Big dtrsmesyrk calls are sent to GPU (inner dimension >= 512)
- Still need to hide data transfer, but there is much less data motion between the host and the device
- Less memory needed on the device, less pinned memory on the host
DTRSM IMPLEMENTATION

- DTRSM is tiled
- Tiles and related copies are submitted to different streams
- Results are kept on the GPU, as they will be needed for dsyrk
- Copy is sent to the CPU
- Host buffers are used for staging host data
DSYRK IMPLEMENTATION

- Dsyrk is tiled
- Different tiles and related copies are submitted to the different streams
- \( L_{21}L_{21}^\dagger \) result is sent to host buffer
- On the host update \( A_{22}^-L_{21}L_{21}^\dagger \) is performed
- \( m=384 \) with 4 streams enough to have GPU occupied
TILED DTRSM/DTRSM/DSYRK

- When inner dimension of dtrsm is too big, it has to be tiled
- $X_1$ is calculated from $X_1A_{11}^T = B_1$ as described before
- dsyrk update is performed if needed
- $B_2 \leftarrow B_2 - X_1A_{21}^T$, $X_1$ is in the GPU memory, $B_2$ and $A_{21}$ are on the CPU
- Process is repeated for the remainder of the matrix
MEMORY REQUIREMENTS

- For tiled dgemm with at most 2048 by 2048 quad-tiles:
  - 12 tiles, 400 MB
- For tiled dtrsm-syrk, with at most 2048 by 2048 tiles:
  - 12 tiles, 400 MB
- Buffer for dtrsm results:
  - With k <= 2048 1 GB is enough for front size up to 61000, bigger fronts can be handled by successive calls to dtrsm and tiled dsyrk
- Total is less than 2 GB
RESULTS

Numerical factorization, Gflops/s

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CPU</th>
<th>Drop-in</th>
<th>High-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>audikw_1</td>
<td>1.48</td>
<td>1.97</td>
<td>2.30</td>
</tr>
<tr>
<td>bone010</td>
<td>1.45</td>
<td>1.93</td>
<td>2.30</td>
</tr>
<tr>
<td>Emilia_923</td>
<td>1.66</td>
<td>1.66</td>
<td>2.30</td>
</tr>
<tr>
<td>Fault_639</td>
<td>1.53</td>
<td>2.23</td>
<td>2.30</td>
</tr>
<tr>
<td>Flan_1565</td>
<td>1.37</td>
<td>1.67</td>
<td>2.36</td>
</tr>
<tr>
<td>Geo_1438</td>
<td>1.67</td>
<td>1.67</td>
<td>2.36</td>
</tr>
<tr>
<td>Hook_1498</td>
<td>1.77</td>
<td>2.01</td>
<td>2.48</td>
</tr>
<tr>
<td>Idoor</td>
<td>0.88</td>
<td>1.12</td>
<td>2.22</td>
</tr>
<tr>
<td>nd24k</td>
<td>1.52</td>
<td>1.52</td>
<td>1.82</td>
</tr>
<tr>
<td>Serena</td>
<td>2.48</td>
<td>2.48</td>
<td>2.48</td>
</tr>
</tbody>
</table>

2 x Xeon E5-2690 v2 + K40 (max boost, ECC=on)
## PROFILE

### Drop-in

<table>
<thead>
<tr>
<th>Process</th>
<th>Tesla K40m</th>
<th>Context 1 (CUDA)</th>
<th>MemCpy (HtoD)</th>
<th>MemCpy (DtoH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;nt24k.bin&quot; (27856)</td>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
<td>[3]</td>
</tr>
</tbody>
</table>

### High-level API

<table>
<thead>
<tr>
<th>Process</th>
<th>Tesla K40m</th>
<th>Context 1 (CUDA)</th>
<th>MemCpy (HtoD)</th>
<th>MemCpy (DtoH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;nt24k.bin&quot; (13932)</td>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
<td>[3]</td>
</tr>
</tbody>
</table>

Compute
HIGH LEVEL INTERFACE FOR LDL FACTORIZATION

- WSMP signals what data is likely to be reused, and it is cached on the GPU.
LDLᵀ FACTORIZATION

1) $L_{11}$

2) $S_{21} = \begin{pmatrix} L_{21} \\ \end{pmatrix}$

3) $C = C - S_{21} L_{21}$

$L_{21}$ and $S_{21}$ are kept in the GPU memory between operations.
### LDL RESULTS

**Numerical Factorization, GFlop/s**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LDL CPU</th>
<th>LDL GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>audkw_1</td>
<td>1.70</td>
<td></td>
</tr>
<tr>
<td>bone010</td>
<td>1.60</td>
<td></td>
</tr>
<tr>
<td>Emilia_922</td>
<td>2.03</td>
<td>2.25</td>
</tr>
<tr>
<td>Fault_539</td>
<td>2.12</td>
<td>2.21</td>
</tr>
<tr>
<td>Flan_1565</td>
<td>1.30</td>
<td></td>
</tr>
<tr>
<td>Geo_1438</td>
<td>1.94</td>
<td></td>
</tr>
<tr>
<td>Hook_1408</td>
<td>1.40</td>
<td></td>
</tr>
<tr>
<td>Idoor</td>
<td>1.36</td>
<td></td>
</tr>
<tr>
<td>nd24k</td>
<td></td>
<td>2.25</td>
</tr>
<tr>
<td>serena</td>
<td></td>
<td>2.21</td>
</tr>
</tbody>
</table>

**Configuration:**

2 x Xeon E5-2690 v2 + K40 (max boost, ECC=on)
SPEEDUP

Fraction of factorization flops with $k \geq 512$
DISTRIBUTED MEMORY PARALLEL FACTORIZATION (MPI)

- Factorization by p processes
- Best performance when p is a power of 2
- Factorization of levels below top log p levels is performed by the processes independently
- Processes cooperate on factoring upper levels of the elimination tree
DISTRIBUTED MEMORY PARALLEL FACTORIZATION

- Update matrices are distributed between processes working on them in a block-cyclic fashion.
- Smaller block size provides better load balancing.
- With bigger block size, efficiency of BLAS3 operations increases.
DISTRIBUTED MEMORY PARALLEL FACTORIZATION

- On the GPU block size ideally should be 512 or more
  - Determined by PCIe and host memcpy copy speed
- Large tiles hurt load balancing
- For our test system, 512 block size provides best performance
CACHING FOR MPI

- **Dtrsm**
  - results are kept in the GPU cache

- **Dgemm**
  - One of the matrices that is going to be reused is put in the GPU cache in the course of performing dgemm operation
  - One of the matrices are in the GPU cache, another matrix is not reused

- **Dsyrk**
  - Matrix can be in the GPU cache from previous operations, or on the host
MPI SCALING RESULTS

Performance, GFlops/s

# of mpi ranks, 10 cores and 1 GPU per rank

- Serena CPU
- Serena GPU
- Audi CPU
- Audi GPU
MPI SCALING ON BLUE WATERS

Factorization Performance, GB/s

Number of cpu cores

Blue Waters, AMD Interlagos, 16 CPU cores vs. 8 CPU cores + 1 K20 GPU
FUTURE WORK

- Share the work with CPU - now for many models it is idle as almost all work is sent to the GPU
- Multi-GPU
- Tuning to automatically set off-load cutoffs for a variety of systems
ACKNOWLEDGEMENTS

- The Private Sector Program at NCSA
- Blue Waters project, supported by NSF (award number OCI 07-25070) and the state of Illinois