GTC 2013:

DEVELOPMENTS IN GPU-ACCELERATED SPARSE LINEAR ALGEBRA ALGORITHMS

Kyle Spagnoli
Research Engineer @ EM Photonics

3/20/2013
INTRODUCTION

» Sparse systems
» Iterative solvers
  » High level benchmarks
» Approximate inverse preconditioner
  » GPU implementation details
  » In-depth benchmarks
» Direct solvers
» EM Photonics’ CULA Sparse library
PREVIOUS WORK – CULA DENSE

» Dense “LAPACK” for CUDA
  » System solvers
  » Least squares solvers
  » Eigenvalues
  » Singular value decomposition

» Multi-GPU support
» 3-7x speedup vs optimized CPU
» Presented at GTC past few years
Sparse System Introduction

» Physical phenomena modeled by partial differential equations

\[ \rho \left( \frac{\partial v}{\partial t} + v \cdot \nabla v \right) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}, \]

» Forces, momentums, velocities, energy, temperature, etc.

» Sparse systems appear in discretization methods

» Finite element, finite volume, topology methods, etc.

» Fluid & structural dynamics, electromagnetics, etc.
Sparse System Introduction

Images courtesy of "Iterative Methods for Sparse Linear Systems" by Yousef Saad
ITERATIVE SOLVERS
ITERATIVE SOLVERS - BASICS

» Want to solve for $x$ in $Ax = b$

» Each iteration works toward lowering the residual value of $r = \|Ax - b\|$

» Krylov subspace methods
  » $K_m(A, v) \equiv \text{span} \{v, Av, A^2v, ..., A^{m-1}v\}$

» Sparse matrix-vector multiplication

» CG, GMRES, BiCGSTAB, Minres, etc.
Iterative Solvers – CG Method

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$

for $i = 1, 2, \ldots$

solve $Mz^{(i-1)} = r^{(i-1)}$

if $i = 1$
    $p^{(1)} = z^{(0)}$
else
    $\beta_i = \rho_i / \rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_i p^{(i-1)}$
endif

$q^{(i)} = Ap^{(i)}$

$\alpha_i = \rho_i / p^{(i)^T} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence; continue if necessary

Apply preconditioner $M$ with a matrix-vector solve

Matrix-vector multiplication
Iterative Solvers - Preconditioners

» Modify original system to make it “easier” to solve

» \( Ax = b \rightarrow MAx = Mb \)

» (Hopefully) reduce the number of iterations needed to solve

» One time setup & generation cost

» Per-iteration application cost
PRECONDITIONERS — JACOBI

» Jacobi / Block Jacobi
  » \( M = \text{diag}(A) \)
  » \( M = \text{block}\_\text{diag}(A, bs) \)

» Minor reduction in iterations

» Lightweight and parallel in generation and application
Zero fill incomplete LU (ilu0)

- \( M \rightarrow LU \approx A \)
- \( M \) has same pattern as \( A \)

Significant reduction in iterations

Performance depends on matrix structure
- Parallelism from graph coloring

Applied with \textit{sparse triangular solve}
- Limited performance on GPU
Preconditioners — Approximate Inverse

» Approximate Inverse
  » \( M^{-1} \rightarrow AM^{-1} \approx I \)

» Factorized Approximate Inverse (for SPD systems)
  » \( M^{-1} \rightarrow LL^T \approx A \)

» \( M^{-1} \) has pattern of \( A \) (or \( A^2, A^3 \))

» Moderate reduction in iterations compared to ILU0

» Very expensive to generate but highly parallel

» Applied with **sparse matrix-vector multiplication**
  » Extremely high performance on GPU
circuit simulation problem with 7.5M non-zero elements
Preconditioner Performance

Residual over time - CPU (multi-core)

Circuit simulation problem with 7.5M non-zero elements
circuit simulation problem with 7.5M non-zero elements
APPROXIMATE INVERSE DETAILS
Factorized Approximate Inverse — Math Lite

» Minimization of Frobenius norm:

$$\|AM - I\|_F^2 \rightarrow \sum_{k=1}^{n} \|\hat{A}_k \hat{m}_k - e_k\|_2^2$$

» Can be solved with per-column parallelism!

» Finding $\hat{A}_k$ is memory intensive...
   » extract a dense small matrix relative to the number of elements in the column

» Minimization is compute intensive...
   » positive definite solve on the small dense matrix
   » least squares solve for non-SPD inputs
» High performance hybrid CPU/GPU solution...
  » Find chunk of successive $\hat{A}_k$ matrices using CPU
    » batched with OpenMP
  » Perform positive definite solves on the GPU
    » batched with custom CUDA kernels
  » Overlap chunks using asynchronous operations

<table>
<thead>
<tr>
<th>CPU</th>
<th>Find $A_{n/4}$</th>
<th>Find $A_{n/4}$</th>
<th>Find $A_{n/4}$</th>
<th>Find $A_{n/4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>Solve $M_{n/4}$</td>
<td>Solve $M_{n/4}$</td>
<td>Solve $M_{n/4}$</td>
<td>Solve $M_{n/4}$</td>
</tr>
</tbody>
</table>
**Factorized Approximate Inverse – Test Matrices**

<table>
<thead>
<tr>
<th>Matrix Name</th>
<th>Description</th>
<th>n</th>
<th>nnz</th>
</tr>
</thead>
<tbody>
<tr>
<td>G3_circuit</td>
<td>circuit simulation</td>
<td>1.5M</td>
<td>7.6M</td>
</tr>
<tr>
<td>apache2</td>
<td>structure</td>
<td>700K</td>
<td>4.8M</td>
</tr>
<tr>
<td>msdoor</td>
<td>structure</td>
<td>400K</td>
<td>19M</td>
</tr>
<tr>
<td>ecology2</td>
<td>2d/3d problem</td>
<td>1M</td>
<td>5M</td>
</tr>
<tr>
<td>offshore</td>
<td>electromagnetics</td>
<td>250k</td>
<td>4.2M</td>
</tr>
<tr>
<td>audikw_1</td>
<td>structure</td>
<td>950k</td>
<td>77M</td>
</tr>
<tr>
<td>ldoor</td>
<td>structure</td>
<td>900k</td>
<td>42M</td>
</tr>
<tr>
<td>thermal2</td>
<td>heat transfer</td>
<td>1.2M</td>
<td>8.5M</td>
</tr>
</tbody>
</table>

Test matrices from the UF Sparse Matrix Collection
FACTORIZED APPROXIMATE INVERSE – BENCHMARKS

GPU: NVIDIA C2070
CPU: Dual Intel Xeon 5660

<table>
<thead>
<tr>
<th>name</th>
<th>speed up*</th>
</tr>
</thead>
<tbody>
<tr>
<td>G3_circuit</td>
<td>9.92x</td>
</tr>
<tr>
<td>apache2</td>
<td>7.14x</td>
</tr>
<tr>
<td>msdoor</td>
<td>11.06x</td>
</tr>
<tr>
<td>ecology2</td>
<td>10.73x</td>
</tr>
<tr>
<td>offshore</td>
<td>8.21x</td>
</tr>
<tr>
<td>audikw_1</td>
<td>8.08x</td>
</tr>
<tr>
<td>ldoor</td>
<td>10.73x</td>
</tr>
<tr>
<td>thermal2</td>
<td>15.00x</td>
</tr>
</tbody>
</table>

* Best CUDA algorithm relative to best CPU algorithm. Includes data transfers, preconditioner generation, and iterations.

did not converge in reasonable time....

Copyright © 2013 EM Photonics
Preconditioner Generation Time Relative CPU ilu0

smaller is better
» Successful in reducing number of iterations
» Very expensive to generate
» Massively parallel
» Excellent preconditioners for GPU
» Chose an algorithm that fits your platform
DIRECT SPARSE LINEAR SYSTEM SOLVERS
DIRECT SPARSE SOLVERS

- LU, Cholesky, or QR factorization of matrix
- Dense methods require full $M \times N$ matrix
- Express problem as a number of relatively small dense problems with task and data parallelism
- Factorize stages:
  1. Symbolic – Analyze graph structure to determine a “plan of attack” that best suits platform
  2. Numeric – Perform factorization calculations
The nonzero pattern forms a elimination tree describing an order of column (vector) operations.

Reordering allows for better parallelism and data locality.
Direct Sparse Solvers - Symbolic Factorization

» Merge adjacent nodes into supernodes that can now utilize high-performance matrix operations

» Fill in with explicit zeros if needed

» Make choices based on the hardware available
  » Zero fill thresholds
  » Number of processors/GPUs
DIRECT SPARSE SOLVERS

» Tree can now expressed as a *task dependency graph* of *dense matrices* operations
  » matrix multiply, triangular solve, small dense factorization

» Traverse graph queuing operations on the GPU

» Process smaller problems on the host

» Manage memory locality to prevent ping-ponging

» Kepler’s **Hyper-Q** improves performance of processing different sized matrices performing different tasks
» Software package available from www.culatools.com
  » Free for personal academic usage
  » Version “S4” available now
  » Version “S5” beta starting in early April
» Platforms
  » Multi-threaded host, CUDA, CUDA (device pointers)

» Solvers
  » CG, BiCG, BiCGStab, BiCGStab(L), GMRES, Minres, ...

» Preconditioners:
  » Jacobi, Block Jacobi, SSOR, ILU0, Ainv, Fainv, ...

» Formats
  » Coo, Csr, Csc, Eli, Dense, ...
// create plan
culaSparsePlan plan;
culSparseCreatePlan(&plan);

// configure plan
culaSparseSetCudaPlatform(plan, &cudaOptions);
culaSparseSetDCooFormat(plan, &cooOptions, n, nnz, a, b, x);
culaSparseSetFainvPreconditioner(plan, &fainvOptions);
culaSparseSetCgSolver(plan, &cgOptions);

// execute plan
culaSparseExecute(plan, &config, &results);
// change preconditioner to ilu0
culaSparseSetIlu0Solver(plan, &ilu0Options);

// re-execute plan with cached data (avoids data transfer)
culaSparseExecute(plan, &config, &results);

// change solver to bicg
culaSparseSetBicgSolver(plan, &bicgOptions);

// re-execute plan with cached data (avoid preconditioner regeneration)
culaSparseExecute(plan, &config, &results);
» “Matrix free” methods
» Multi-GPU support
» More solvers, preconditioners, data formats, and performance
» PETSc integration
» Intel Xeon Phi platform
» Direct solver releases
Thanks!
Questions?

» Convention hall @ booth 311
» More information @ www.culatools.com