Improving Gaussian Elimination on Multi-Core Systems

Marwa A. Al–Shandawely
PDC/KTH
Agenda

- Algorithm overview.
- Trivial parallelization.
- Problems.
- Sequential optimization
- Proposed solutions.
- Experimental results.
- Conclusions and future work.
for i=1 to n-1
  find pivotPos in column i
  if pivotPos ≠ i
    exchange rows(pivotPos,i)
  end if

for j=i+1 to n
  A(i,j) = A(i,j)/A(i,i)
end for j

!$omp parallel do private ( i ,j )
for j=i+1 to n+1
  for k=i+1 to n
    A(k,j)=A(k,j)-A(k,i)×A(i,j)
  end for k
end for j
end for i
Trivial OMP Speed up

N=1000  N=2000  N=3000  N=4000  N=5000
Problems

- Poor data locality
- Pivoting is done by master thread
- Overheads of creating and destroying threads at each iteration
- Sequential optimization
Sequential optimization

- Replace division by the constant pivot
- Avoid loop invariant access in the inner most loop
- Eliminate the check for pivot changing position
- Make use of fortran array notation

\[
\text{Do } k=j+1,n \\
A(k,j) = A(k,j) / A(j,j) \\
\text{End do}
\]

\[
C = 1 / A(j,j) \\
A(j+1:n) = A(j+1:n) * c
\]
Pivots array

Locks array

Pivot holder

- Eliminate (i) on column(i+1)
- Search (i+1)
- Store pivot (i+1) position
- Prepare colmn (i+1)
- Free lock (i+1)
- Eliminate (i) on rest of scope
Cyclic Column distribution speed up
What went wrong?!

- The original algorithm requires pivot columns to be prepared in order while the whole matrix is accessed for each pivot column.

- For large input sizes; the cache is evicted many times for each iteration and there is no reuse of data in the cache.

- False sharing on pivots and locks array.
Improving!

- Double elimination on pivot holders.
  - Knowledge of two pivots allow data reuse.

- Each column is an accumulation of eliminations using previous columns!
  - Make more pivots available each step and eliminate each column using several pivots while it is in the cache.
Cyclic Block distribution

- Block of pivots
- Increase work/iter.
- Increase locality
- Less locks
- Load balancing?!
Cyclic Block distribution

- Block of pivots
- Increase work/iter.
- Increase locality
- Less locks
- Load balancing?!
Speed up for block distribution

N=2000

N=5000

C=1
C=2
C=3
C=4
C=5

Original

N=2000

N=5000
Speed up as $C=25$

N=2000

N=5000

Original

double elimination

C=25 with double elimination
Conclusions and future work

- Scalable performance on multicores is highly dependent on application implementation, data layout and access patterns.

- Cache and memory access optimization techniques is vital for performance despite the loss of readability.

- Future work:
  - Adaptive blocking scheme that changes the block size as a function of the matrix size, cache settings, and number of cores.
Questions