Collaborators Over the Years

Stanford
Bob Dutton (advisor) & Tom Blank (now Microsoft)
Jerry Tiemann (GE CRD) & Arthur Raefsky (now SGI)

IDA
John Conroy, Steve Kratzer & Aaron Naiman (now JCT)

LSTC
Roger Grimes & Cleve Ashcraft

Many discussions with others
Golub, Duff, Simon, Amestoy, etc.
The linear solver is a major computational bottleneck in Mechanical Computer Aided Engineering (MCAE)

Multifrontal method is used in:
• NASTRAN – Vibration
• ANSYS – Linear Analysis
• ABAQUS – Implicit Non-Linear
• LS-DYNA – Explicit Non-Linear
Outline

Review of Multifrontal Method
Distributed Memory
Shared Memory
SIMD
Petascale?
A matrix is sparse if it has a non-zero structure that can be exploited to reduce storage and/or operations.
Iain Duff’s definition of sparsity.
Sparsity and Fill-in Example

O(N) factor operations

1  X     X
2  X     X
3  X     X
4  X     X
5  X     X
6  X     X
7  X     X
8  X     X
9  X     X

O(N^3) factor operations

9  XXXXXXXXXXXX
1  XXXXXXXXXXXX
2  XXXXXXXXXXXX
3  XXXXXXXXXXXX
4  XXXXXXXXXXXX
5  XXXXXXXXXXXX
6  XXXXXXXXXXXX
7  XXXXXXXXXXXX
8  XXXXXXXXXXXX
do 4 k = 1, 9
  do 1 i = k + 1, 9
    a(i, k) = a(i,k) / a(k,k)
    continue
  do 3 j = k + 1, 9
    do 2 i = k + 1, 9
      a(i,j) = a(i,j) - a(i,k) * a(k,j)
      continue
    continue
  continue
  continue
  continue
Multifrontal View of Toy

Duff and Reid, ACM TOMS 1983
Multifrontal Attributes

Exploits fill-reducing orderings
  (i.e., METIS, Multisection, or MMD)

Dense arithmetic kernels
  • Matrix-matrix operations
  • High performance for large frontal matrices

Easily adapts for pivoting
  • Dynamically form frontal matrices
  • Allows symmetric indefinite problems

Relatively small working set
  • Good for hierarchical memories
  • Out-of-core
Toy Problem (with stack)

Post-order traversal of Elimination Tree

- Minimizes working storage (Liu ’85)
- Limits concurrency
do 1 sn = 1, nsn
   call Assemble()
   call Factor()
   call Stack()
1     continue
In-Core Memory Trace

Updates, working
Factor, working
Updates, stacked
Factor, complete

Stack
Total Memory

Time
A Real Problem: “Hood”

Automotive Hood Inner Panel
Springback using LS-DYNA
“Hood” Elimination Tree

Each frontal matrix’s triangle scaled by operations required to factor it.
Hood Storage Trace with Stack

HOOD: total storage for post-order traversal

(factor storage + working storage) / total factor storage

working storage

factor entries

# factor ops / total factor ops
Outline

Review of Multifrontal Method
Distributed Memory
Shared Memory
SIMD
Petascale?
Design Constraints

Practical problems run on O(10) CPUs
ANSYS runs on up to 8
Toyota runs LS-DYNA on 32
Target platform is the ubiquitous Beowulf
implies MPI-1
want to minimize communication
Integrate with LS-DYNA code
suggests f77
Two Sources of Concurrency

Concurrency within frontal matrices
- Small $P$ => column wrap
- Large $P$ => 2D (ala LINPACK benchmark)

Concurrency across elimination tree
- Frontal matrices only dependent on children
- “Subtree – subcube” typically used
- Limits communication
Subtree-Subcube Distribution

Level 1

Level 2

Level 3

Processor 0

Processor 1
do 1 sn = 1, nsn
   if ((type(sn) .eq. sequential) .and.
   1      (owner(sn) .eq. mid)) then
      call Assemble()
      call Factor()
   else if ((type(sn) .eq. parallel) .and.
   1           (member(sn) .eq. .true.)) then
      call MPP_Assemble()
      call MPP_Factor()
   end if
   call Stack()
   1   continue
do 2 k = 1, sn_size, 60
   if (panel_owner(k) .eq. mid) then
      call Seq_Factor_Panel()
   endif
   call MPI_Bcast(*,*,*,panel_owner(k),*,*)
do 1 j = k + 60, sn_ld, 60
   if (panel_owner(j) .eq. mid) then
      call Seq_Panel_Update()
   endif
1 continue
2 continue
Dense solver on Origin 2000

MFlop/s

processors

- without pivoting
- with pivoting
1D Dense Kernel Performance

Dense solver on Cray X1

- GFlop/s vs. MSPs
- Two curves: 9801 (blue) and 19503 (pink)
- Performance improves as the number of MSPs increases
Hood Performance

Hood Factorization on O2k

Hood Solves on O2k

LS-DYNA Oct 2002
26.5 GFlops to factor
9.1 GFlop/s, 36% peak
DSG Performance

DSG factorization on O2k

DSG solves on O2k

ANSYS Nov 2002
486 GFlops to factor (i4r8)
25 sec. symbolic processing
7 sec. matrix distribution
DSG Speedup on O2k

- Measured speedup
- Modeled speedup

Graph showing speedup versus number of processors.
Static Load Balance Problem

- Subtree – Subcube is static mapping
  - Good - reduces communication
  - Bad - freezes load imbalance
    - Can’t get it exactly right
    - Graph partitioning is NP-complete!
- Load imbalance shows up in speedup models
  - Model is simple
    - Floating point ops per processor
    - MPP matrices divided evenly
    - No communication
  - DSG had no hope of achieving 25/32
Sony factorization on O2k

Sony solves on O2k

LS-DYNA Jan 2003
486 GFlops to factor (i4r4)
52 sec. symbolic processing
9 sec. matrix distribution
Sony Factor Speedup

Sony speedup on O2k

- [Graph showing speedup vs. number of processors]
  - Speedup measured (blue squares)
  - Speedup modeled (pink squares)
Sony Memory Scaling

Numeric storage on PE 0

MWords

processors

num on PE 0
Difficult to visualize or understand impact of load imbalance or synchronization

Did I make the right trade-offs?

Debugging MPI code is extremely difficult

e.g., LAM 7 and 64-bit integers

Performance portability increasingly hard

DAXPY vs. DDOT kernels?

Unrolling and block sizes?
Multifrontal factorization and solve kernels dominate the run time of block-shift-invert Lanczos code.
Static Load Balance Problem

Eigen solver time balance is ideally
- 1/3 factorization
- 1/3 solves
- 1/3 DGEMM

Optimal distribution for each phase is different!
- I use distribution for factorization and accept imbalance
- Alternative is major communication overhead
Outline

Review of Multifrontal Method
Distributed Memory
Shared Memory
SIMD
Petascale?
Design Constraints

Practical problems run on O(10) CPUs
- target is server-scale SMP
- Enhance LS-DYNA SMP code
  - implies f77
- Goal is to maximize concurrency w/o major change to code
  - use OpenMP
Breadth-First Traversal
do 3 level = leaves, root
   if (sn_count(level) .gt. lg(npe)) then
     c$omp paralleldo
       do 1 sn = ptr(level), ptr(level + 1) - 1
       call Seq_Assemble()
       call Seq_Factor()
     1      continue
   else
     do 2 sn = ptr(level), ptr(level + 1) - 1
     call Seq_Assemble()
     call SMP_Factor()
   2      continue
   end if
   call Storage_Recovery()
continue
Notional SMP_Factor Kernel

do 2 k = 1, sn_size, 64
   call Seq_Factor_Panel()
do 1 j = k + 64, sn_ld, 64
   c$dir paralleldo
      call Seq_Panel_Update()
1      continue
2      continue
### Breadth-first Memory Trace

<table>
<thead>
<tr>
<th>Levels eliminated</th>
<th>Memory used</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Levels eliminated" /></td>
<td><img src="image2" alt="Memory used" /></td>
</tr>
</tbody>
</table>

**Updates**

- Factor, working
- Factor, done

<table>
<thead>
<tr>
<th>Updates</th>
<th>Memory used</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Updates" /></td>
<td><img src="image4" alt="Memory used" /></td>
</tr>
</tbody>
</table>

**USC School of Engineering**

**ISI Information Sciences Institute**
Why?

As we ascend the elimination tree:
  Number of Frontal Matrices per level halves
  Size of each frontal matrix roughly doubles

Net result:  Working storage nearly constant
## Test Problems

All extracted from LS-DYNA applications. All use METIS ND ordering (thanks George!)

| Matrix | Equations | $|A|$   | $|L|$          | Gops  |
|--------|-----------|--------|---------------|-------|
| Hood   | 235962    | 6511613| 52707137      | 26.11 |
| I-Beam | 615600    | 16845804| 152999928     | 110.39|
| Knee   | 69502     | 2611222| 40059241      | 41.07 |
Factorization Time on O2K

The graph shows the log10(CPU time) on the y-axis and the number of processors on the x-axis. It compares the measured and ideal factorization CPU times.
Hood Performance on O2K
Factorization Speedup on O2K

HOOD: factorization speedup

speedup

# of processors

ideal

measured
Hood Speedup on O2K
## Comparison with MPI version

<table>
<thead>
<tr>
<th>Metric</th>
<th>Stackless</th>
<th>Subtree-subcube</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Complexity</td>
<td>Simple</td>
<td>Difficult</td>
</tr>
<tr>
<td>Work Assignment</td>
<td>Dynamic</td>
<td>Static</td>
</tr>
<tr>
<td>Storage Management</td>
<td>Simple</td>
<td>Non-trivial</td>
</tr>
<tr>
<td>Ordering Sensitivity</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>
Easy SMP parallelism:
• Early results generated with two parallel DOALL’s
Arbitrary number of processors
• Not limited to powers of two
SMP can adapt dynamically
• Storage Management
• Work Assignment
Maximizing concurrency has a price 😞

Working Set comparison:
• Breadth-first: $0.75 \times |L|$  
• Post-order: $0.10 \times |L|$  

Can lead to TLB thrashing  
• Not just caches to worry about
Looking Backwards

This was a great idea 😊
Generating concurrency was easy
Debugging performance was impossible
Why *only* getting 12/16 at mid-level?
What was wrong with root SNs?
Page thrashing?
Not even FlashPoint helped
Recursive DOALLs would have helped
Outline

- Review of Multifrontal Method
- Distributed Memory
- Shared Memory
- SIMD
- Petascale?
“Massively Parallel”
• DEC Alpha workstation host
• 2D array of 4-bit processors
  • 1k – 16k
• 2D interconnection network
  • Nearest neighbor
  • Broadcasts
  • High bandwidth
• 2-stage switch network
  • Arbitrary links
  • Low bandwidth
Design Constraints

SIMD programming model

Research project so I had no other constraint!!!

Goal was to demonstrate whether or not it was even possible to get good SIMD performance for a sparse matrix factorization
2D “planes” of processors
2D covering of frontal matrix
=> 4D distribution of matrix

Types of operations:
• Factor diagonal plane
• Off-diagonal elimination of L and U
• Matrix-matrix multiply to compute Schur complement
for (sn = 1; sn <= nsn; ++ sn)
{
    SIMD_Assemble();
    for (k = 1; k <= sn_size; k += sqrt(npe))
    {
        SIMD_invert_diagonal_plane();
        Spread_Y();
        for (i = k + sqrt(npe); i < sn_ld; i += sqrt(npe))
            SIMD_eliminate_plane();
        Spread_X();
        for (j = k + sqrt(npe); j < sn_ld; j += sqrt(npe))
            for (i = k + sqrt(npe); i < sn_ld; i += sqrt(npe))
                SIMD_DGEMM();
    }
    SIMD_Stack();
}
Dense Kernel Performance

![Graph showing the performance of different systems over varying matrix sizes.](image)

- **SPEED (MFLOPS)**
- **MATRIX SIZE**
- **CM-5 (16 nodes)**
- **MP-2 (16K)**
- **C90**
Dense Time Breakdown

TIME (SEC)

MATRIX SIZE

BK SOLVE
SCHUR COMPL.
FWD SOLVES
DIAGONALS
Pin matrix rows/columns to processor array, introducing zeroes as necessary
Only communicate with nearest neighbors
SIMD Performance

- 100x100 GRID
- BCSSTK-23
- BCSSTK-33
- 30x30x30 GRID
- 35x35x35 GRID

Speed (Mflops)

- CM-5 (16 nodes)
- MP-2 (16K procs)
- C-90
SIMD Scaling

The graph shows the scaling of different computations (BK SOLVE, REMOTE, LOCAL, SCHUR COMPL, FWD SOLVES, and DIAGONALS) with respect to the number of Processing Elements (PEs). The x-axis represents the number of PEs, ranging from 1024 to 16384. The y-axis represents the time in seconds, ranging from 0 to 6.

- BK SOLVE: The red line shows the time for BK SOLVE scaling.
- REMOTE: The black line shows the time for REMOTE scaling.
- LOCAL: The gray line shows the time for LOCAL scaling.
- SCHUR COMPL: The dark gray line shows the time for SCHUR COMPL scaling.
- FWD SOLVES: The light gray line shows the time for FWD SOLVES scaling.
- DIAGONALS: The white line shows the time for DIAGONALS scaling.

As the number of PEs increases, the time for all computations decreases, indicating efficient scaling.
SIMD Results Mixed

Achieved 2/3 peak for 3D matrices. Stationary mapping cost a lot of memory.

Never tried to compress

Individual solves relatively slow.

SIMD reordering only briefly examined.

Approximate MMD

No work on symbolic preprocessing.
Looking Backwards

High Performance Fortran (HPF) dialects were good for organizing data and expressing I/O
- MPF (MasPar)
- CMF (Thinking Machines)
Performance programming required
- MPL for MasPar
- AC for CM-5
Outline

Review of Multifrontal Method
Distributed Memory
Shared Memory
SIMD
Petascale?
Current work:
Aimed at O(10) processors
Most work is in sequential subtrees
Load imbalance preferred vs. communication
Sequential ordering and symbolic tolerable
Successfully deployed
Near-term O(100) processors

Evolve from present code
Add 2D frontal matrices near root of tree
Duff and Amestoy
IBM (Watson & Austin)
Insist on tighter load balance
Can’t afford Amdahl fractions
Accept additional communication required
Drop out-of-core?
Out-of-core dense kernels are slow
Working set ~ |L| anyway
Long-term (Petascale)

Very large matrices
  $10^9$ degrees of freedom vs. $10^7$ today
In-core thanks to Petascale memory
Number of processors will exceed SNs.
  Smallest fronts will have 10-100 processors
    There will be no sequential subtrees
    Only support 2D distribution
Need global address space to minimize synchronization and messaging overhead
Breadth-first to maximize concurrency
Petascale Breadth-first

Level 1

Level 2

Level 3

All SNs 2D
Adapt to each level
Global address space
=> no stationary map
Bottom Line:
Blend ideas from Stackless and SIMD
MPI code is a red herring
Research questions:
How does one reorder?
ParMETIS?
Parallel symbolic factorization?
No prior art to my knowledge