Tuning Sparse Matrix Vector Multiplication for multi-core SMPs
(paper to appear at SC07)

Sam Williams
samw@cs.berkeley.edu
Other Authors
(for the SC07 Paper)

• Rich Vuduc
• Lenny Oliker
• John Shalf
• Kathy Yelick
• Jim Demmel
Outline

• Background
  – SpMV
  – OSKI
• Test Suite
  – Matrices
  – Systems
• Results
  – Naïve performance
  – Performance with tuning/optimizations
• Comments
Background
Dense Matrix Vector Multiplication

• Evaluate $y = Ax$
• $x$ & $y$ are dense vectors
• $A$ is a dense matrix
• Each element is required to access the source vector ($X$) = trivial address calculation
• Each row is required to access the destination vector ($Y$) = trivial address calculation
• Trivial to exploit ILP, DLP, TLP
• Friendly access to the vectors
• Low computational intensity - likely memory bound
Sparse Matrix Vector Multiplication

• Evaluate $y = Ax$
• $x$ & $y$ are still dense vectors
• $A$ is a sparse matrix
• Unlike a dense matrix, only the nonzeros are stored and operated on.
• Unlike dense matrix multiplication, significant meta data is required to tag the coordinates of each nonzero.
• Nonzeros normally laid out in rows (CSR)
• Difficult to exploit ILP, DLP, TLP
• Unfriendly access to the source vector
• Even lower computational intensity
  - likely to be heavily memory bound
OSKI & PETSc

• Register Blocking reorganizes the matrix into tiles by adding nonzeros
• better ILP/DLP at the potential expense of extra memory traffic.

• OSKI is a serial auto-tuning library for sparse matrix operations developed at UCB
• OSKI is primarily focused on searching for the optimal register blocking

• For parallelism, it can be included in the PETSc parallel library using a shared memory version of MPICH
• We include these 2 configurations as a baseline comparison for the x86 machines.
Exhaustive search in the multi-core world?

• Search space is increasing rapidly (register/cache/TLB blocking, BCSR/BCOO, loop structure, data size, parallelization, prefetching, etc…)

• Seemingly intractable

• Pick your battles:
  – use heuristics when you feel confident you can predict the benefit
  – search when you can’t.
Test Suite
Sparse Matrices

2K x 2K Dense matrix stored in sparse format

Well Structured (sorted by nonzeros/row)
- Protein
- FEM / Spheres
- FEM / Cantilever
- Wind Tunnel
- FEM / Harbor
- QCD
- FEM / Ship
- Economics
- Epidemiology

Poorly Structured hodgepodge
- FEM / Accelerator
- Circuit
- webbase

Extreme Aspect Ratio (linear programming)
- LP

- Pruned original BeBOP suite down to 14
- Subdivided them into 4 categories
- None should fit in an Opteron’s cache.
- Rank ranges from 2K to 1M
Multi-core SMP Systems

Intel Clovertown

AMD Opteron

Sun Niagara2

IBM Cell Blade
Multi-core SMP Systems

**Intel Clovertown**
- Core2/Core2
- 4MB Shared L2
- FSB
- 10.6GB/s
- Blackford Chipset

**AMD Opteron**
- 1MB victim
- Opteron
- 1MB victim
- Opteron
- Memory Controller / HT
- DDR2 DRAM
- 10.6GB/s

**Sun Niagara2**
- 8K D$
- MT UltraSparc
- UltraSparc
- FPU
- EIB (Ring Network)
- 4 FBDIMM memory controllers
- 42.7GB/s (read), 21.3 GB/s (write)

**IBM Cell Blade**
- Very High flop:byte ratio (3.52)
- Moderate flop:byte ratio (0.83) and (0.57)
- Very low flop:byte ratio (0.26)
Multi-core SMP Systems

Intel Clovertown

Conventional Cache-based Memory Hierarchy

AMD Opteron

Disjoint Local Store Memory Hierarchy

Sun Niagara2

IBM Cell Blade
Multi-core SMP Systems

Intel Clovertown

AMD Opteron

Sun Niagara2

IBM Cell Blade
Multi-core SMP Systems

Hardware makes programming easy (?)
Most optimizations on cache-based machines are necessities on Cell.
Speedup over OSKI

- PETSc used to parallelize OSKI

<table>
<thead>
<tr>
<th></th>
<th>1 core</th>
<th>All cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clovertown</td>
<td>1.66x</td>
<td>2.22x</td>
</tr>
<tr>
<td>Opteron</td>
<td>1.44x</td>
<td>3.33x</td>
</tr>
</tbody>
</table>

- New serial optimizations can help some
- Parallelization optimizations are an essential
Naïve Single Thread Performance

Intel Clovertown

~3.6GB/s

AMD Opteron

~3.0GB/s

Sun Niagara2

10x Slower
Parallelization

- Row parallelization based on nonzeros
- Row granularity is a cache line
- Load balancing can be challenging since source vector communication was not included.

- **Exhaustive search** using powers of 2 number of threads
- Explore outward (SMT, multi-core, multi-socket)
Naïve Parallel Performance

Intel Clovertown

2x performance with 8x threads
~6GB/s

1.4x performance with 4x threads
~4.2GB/s

AMD Opteron

40x performance with 64x threads
~16GB/s

Sun Niagara2

~6GB/s

~4.2GB/s

~16GB/s

Naïve all sockets, cores, threads

Naïve Single Thread
Exploit NUMA / Affinity

• Allocate each thread’s work separately.
• Bind each block to that core’s NUMA node.
• Pin processes to respective cores.

• Use routines in Linux, Solaris, and libnuma.
Performance with NUMA / Affinity

Intel Clovertown

AMD Opteron

Sun Niagara2

~6GB/s

2x performance
~9.6GB/s

~16GB/s

+NUMA/Affinity
Naïve all sockets, cores, threads
Naïve Single Thread
Software Prefetching

- Memory Latency is significant
- Memory access pattern is easily expressible
- Hardware prefetchers should be able to infer the patterns and hide the latency
- They cannot infer the temporal locality (none for the matrix)

- Use software prefetch intrinsics
- **Exhaustive search** for the optimal distance
- On Cell, use double buffered DMAs
Performance with SW Prefetch

Intel Clovertown

AMD Opteron

Sun Niagara2

- Benefit of nta hint?
- ~8GB/s
- 1.4x performance
- ~12GB/s
- ~16GB/s

Legend:
- +Software Prefetching
- +NUMA/Affinity
- Naïve all sockets, cores, threads
- Naïve Single Thread
Memory Traffic Minimization Heuristic

• Propose that any machine with enough threads/cores should be memory bound.

• We can do better than just being memory bound - we can try to minimize the memory traffic

• **Heuristic**: select the power of 2 register blocking, CSR/COO format, 16b/32b indices, etc… that minimizes the matrix size.

• **Side effect**: matrix may be minimized to the point where it fits entirely in cache
Code Generation

• Write a Perl script to generate all kernel variants.
• For generic C, x86/Niagara used the same generator
• Separate generator for SSE
• Separate generator for Cell’s SIMD

• Produce a configuration file for each architecture that limits the optimizations that can be made in the data structure, and their requisite kernels
Memory Traffic Minimization Performance

Intel Clovertown

- up to 2x performance
- ~11 GB/s

AMD Opteron

- up to 1.6x performance
- ~13 GB/s

Sun Niagara2

- up to 1.7x performance
- ~19 GB/s

Legend:
- +Memory Traffic Minimization
- +Software Prefetching
- +NUMA/Affinity
- Naïve all sockets, cores, threads
- Naïve Single Thread
Cache and TLB Blocking

- Access to the destination vector is streaming
- Access to the source vector can be random
- Reorganize matrix (and thus access pattern) to maximize reuse.

- **Heuristic**: block destination, then keep adding more columns as long as the number of source vector cache lines (or pages) touched is less than the cache (or TLB). Apply all previous optimizations individually to each cache block.

- **Search**: neither, cache, cache & TLB

- Better locality at the expense of confusing the hardware prefetchers.
- For Cell, express this as a DMA list
Performance with Cache Blocking

Intel Clovertown

AMD Opteron

Sun Niagara2

+Cache/TLB Blocking
+Memory Traffic Minimization
+Software Prefetching
+NUMA/Affinity
Naïve all sockets, cores, threads
Naïve Single Thread
Banks, Ranks, and DIMMs

- As the number of threads increases, so does the number of streams.
- Most memory controllers have finite capability to reorder the requests. (DMA can avoid or minimize this)
- Bank conflicts become increasingly likely
- More DIMMs, configuration of ranks can help
Performance with more DIMMs/Ranks

Intel Clovertown

- ~11GB/s

AMD Opteron

- up to 1.5x from extra DIMMs
- ~14GB/s

Sun Niagara2

- up to 1.3x from firmware fix
- ~23GB/s

IBM Cell Blade

- ~47GB/s
Heap Management with NUMA

- New pages added to the heap can be pinned to specific NUMA nodes (system control)
- However, if that page is ever free()'d and reallocated (library control), then affinity cannot be changed.
- As a result, you shouldn’t free() pages bound under some NUMA policy.
Bandwidth Summary

• The dense matrix is a tool for understanding memory performance

<table>
<thead>
<tr>
<th></th>
<th>GB/s(1P)</th>
<th>GB/s(2P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clovertown</td>
<td>5.4</td>
<td>11.2</td>
</tr>
<tr>
<td>Opteron</td>
<td>7.2</td>
<td>14.0</td>
</tr>
<tr>
<td>Niagara2</td>
<td>23.2</td>
<td>N/A</td>
</tr>
<tr>
<td>Cell</td>
<td>24.7</td>
<td>47.3</td>
</tr>
</tbody>
</table>

• Clovertown gets only 50%, but Kentsfield can get >85%
• 2.2GHz Opteron gets 66%, but 3.0GHz Opteron gets 75%
• Niagara2 gets ~50% of both GB/s and GFlop/s
• Cell gets >90% for virtually every matrix (not just dense)
Comments

- Machines with a large number simple cores performed very well (independent of clock rate)
- Machines with complex cores required significant tuning/optimization
- Niagara2 and Clovertown likely suffer from Little’s law (need sufficient concurrency to cover bandwidth x latency)
- Niagara2 delivers good (single socket) performance and productivity
- Cell delivers good efficiency and performance

- Single thread performance was as good or better performance than OSKI despite using heuristics.
- Parallel Performance was significantly better than PETSc+OSKI
Questions?