Tuning Sparse Matrix Vector Multiplication for multi-core SMPs

Samuel Williams\textsuperscript{1,2}, Richard Vuduc\textsuperscript{3}, Leonid Oliker\textsuperscript{1,2}, John Shalf\textsuperscript{2}, Katherine Yelick\textsuperscript{1,2}, James Demmel\textsuperscript{1,2}

\textsuperscript{1}University of California Berkeley
\textsuperscript{2}Lawrence Berkeley National Laboratory
\textsuperscript{3}Georgia Institute of Technology

samw@cs.berkeley.edu
Multicore is the de facto performance solution for the next decade

Examined Sparse Matrix Vector Multiplication (SpMV) kernel
- Important HPC kernel
- Memory intensive
- Challenging for multicore

Present two autotuned threaded implementations:
- Pthread, cache-based implementation
- Cell local store-based implementation

Benchmarked performance across 4 diverse multicore architectures
- Intel Xeon (Clovertown)
- AMD Opteron
- Sun Niagara2
- IBM Cell Broadband Engine

Compare with leading MPI implementation (PETSc) with an autotuned serial kernel (OSKI)
Sparse Matrix Vector Multiplication

Sparse Matrix
- Most entries are 0.0
- Performance advantage in only storing/operating on the nonzeros
- Requires significant meta data

Evaluate $y = Ax$
- $A$ is a sparse matrix
- $x$ & $y$ are dense vectors

Challenges
- Difficult to exploit ILP (bad for superscalar),
- Difficult to exploit DLP (bad for SIMD)
- Irregular memory access to source vector
- Difficult to load balance
- **Very low computational intensity** (often >6 bytes/flop)
Test Suite

- Dataset (Matrices)
- Multicore SMPs
Matrices Used

- Pruned original SPARSITY suite down to 14
- none should fit in cache
- Subdivided them into 4 categories
- Rank ranges from 2K to 1M
Multicore SMP Systems

Intel Clovertown

AMD Opteron

Sun Niagara2

IBM Cell Blade
Multicore SMP Systems
(memory hierarchy)

Intel Clovertown

AMD Opteron

Sun Niagara2

IBM Cell Blade

Conventional Cache-based Memory Hierarchy

Disjoint Local Store Memory Hierarchy
Multicore SMP Systems
(cache)

Intel Clovertown

AMD Opteron

Sun Niagara2

IBM Cell Blade
Multicore SMP Systems
(peak flops)

Intel Clovertown

AMD Opteron

Sun Niagara2

IBM Cell Blade
Multicore SMP Systems
(peak read bandwidth)

Intel Clovertown

AMD Opteron

Sun Niagara2

IBM Cell Blade

Fully Buffered DRAM
Multicore SMP Systems (NUMA)

**Intel Clovertown**
- Core2/4MB Shared L2
- FSB
- Chipset (4x64b controllers)
- 10.6GB/s
- 21.3 GB/s (read)
- 10.6 GB/s (write)
- Fully Buffered DRAM

**AMD Opteron**
- 1MB victim
- 1MB victim
- Opteron
- Opteron
- Memory Controller / HT
- 10.6GB/s
- 4GB/s (each direction)
- DDR2 DRAM

**Sun Niagara2**
- FPU MT UltraSparc 8K DS
- 44.96 GB/s (write)
- 44.96 GB/s (read)
- 4GB/s (write)
- 42.7GB/s (read), 21.3 GB/s (write)
- Fully Buffered DRAM

**IBM Cell Blade**
- 512K L2
- PPE
- MFC 256K SPE
- SPE 256K MFC
- 25.6GB/s
- XDR DRAM
- EIB (Ring Network)
- <<20GB/s each direction
- BIF
- XDR
- SPE 256K MFC
- SPE 256K MFC
- SPE 256K MFC
- SPE 256K MFC
- SPE 256K MFC
- SPE 256K MFC
- SPE 256K MFC
- SPE 256K MFC

*Non-Uniform Memory Access*
Naïve Implementation

- For cache-based machines
- Included a median performance number
Vanilla C implementation
- Matrix stored in CSR (compressed sparse row)
- Explored compiler options - only the best is presented here
Pthread Implementation

- Optimized for multicore/threading
- Variety of shared memory programming models are acceptable (not just Pthreads)
- More colors = more optimizations = more work
Matrix partitioned by rows and balanced by the number of nonzeros

SPMD like approach

A barrier() is called before and after the SpMV kernel

Each sub matrix stored separately in CSR

Load balancing can be challenging

# of threads explored in powers of 2 (in paper)
Naïve Parallel Performance

Intel Clovertown

AMD Opteron

Sun Niagara2

Naïve Pthreads
Naïve Single Thread
Naïve Parallel Performance

**Intel Clovertown**

- 8x cores = 1.9x performance

**AMD Opteron**

- 4x cores = 1.5x performance

**Sun Niagara2**

- 64x threads = 41x performance

---

Naïve Pthreads
Naïve Single Thread
Naïve Parallel Performance

Intel Clovertown

1.4% of peak flops
29% of bandwidth

AMD Opteron

4% of peak flops
20% of bandwidth

Sun Niagara2

25% of peak flops
39% of bandwidth

Naïve Pthreads
Naïve Single Thread
Case for Autotuning

- How do we deliver good performance across all these architectures, across all matrices without exhaustively optimizing every combination

- Autotuning
  - Write a Perl script that generates all possible optimizations
  - Heuristically, or exhaustively search the optimizations
  - Existing SpMV solution: OSKI (developed at UCB)

- This work:
  - Optimizations geared for multi-core/-threading
  - generates SSE/SIMD intrinsics, prefetching, loop transformations, alternate data structures, etc…
  - “prototype for parallel OSKI”
Exploiting NUMA, Affinity

- Bandwidth on the Opteron(and Cell) can vary substantially based on placement of data

- Bind each sub matrix and the thread to process it together
- Explored libnuma, Linux, and Solaris routines
- Adjacent blocks bound to adjacent cores
Performance (+NUMA)

Intel Clovertown

AMD Opteron

Sun Niagara2

+NUMA/Affinity
Naïve Pthreads
Naïve Single Thread
Performance (+SW Prefetching)

Intel Clovertown

AMD Opteron

Sun Niagara2

- +Software Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve Single Thread
Matrix Compression

- For memory bound kernels, minimizing memory traffic should maximize performance
- Compress the meta data
  - Exploit structure to eliminate meta data
- **Heuristic**: select the compression that minimizes the matrix size:
  - power of 2 register blocking
  - CSR/COO format
  - 16b/32b indices
  - etc…

- **Side effect**: matrix may be minimized to the point where it fits entirely in cache
- Accesses to the matrix and destination vector are streaming
- But, access to the source vector can be random
- Reorganize matrix (and thus access pattern) to maximize reuse.
- Applies equally to TLB blocking (caching PTEs)

- **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.
Performance (+cache blocking)

Intel Clovertown

AMD Opteron

Sun Niagara2

+Cache/TLB Blocking
+Matrix Compression
+Software Prefetching
+NUMA/Affinity
Naïve Pthreads
Naïve Single Thread
In this SPMD approach, as the number of threads increases, so does the number of concurrent streams to memory.

Most memory controllers have finite capability to reorder the requests. (DMA can avoid or minimize this)

Addressing/Bank conflicts become increasingly likely

Add more DIMMs, configuration of ranks can help

Clovertown system was already fully populated
Performance (more DIMMs, …)

- Intel Clovertown
- AMD Opteron
- Sun Niagara2

Legend:
- +More DIMMs, Rank configuration, etc…
- +Cache/TLB Blocking
- +Matrix Compression
- +Software Prefetching
- +NUMA/Affinity
- Naïve Pthreads
- Naïve Single Thread
Performance (more DIMMs, …)

Intel Clovertown

- 4% of peak flops
- 52% of bandwidth

AMD Opteron

- 20% of peak flops
- 66% of bandwidth

Sun Niagara2

- 52% of peak flops
- 54% of bandwidth

- More DIMMs, Rank configuration, etc...
- Cache/TLB Blocking
- Matrix Compression
- Software Prefetching
- NUMA/Affinity
- Naïve Pthreads
- Naïve Single Thread
Performance (more DIMMs, …)

Intel Clovertown

3 essential optimizations

AMD Opteron

4 essential optimizations

Sun Niagara2

2 essential optimizations

+More DIMMs, Rank configuration, etc…
+Cache/TLB Blocking
+Matrix Compression
+Software Prefetching
+NUMA/Affinity
Naïve Pthreads
Naïve Single Thread
Cell Implementation

- Comments
- Performance
No vanilla C implementation (aside from the PPE)

Even SIMDized double precision is extremely weak
- Scalar double precision is unbearable
- Minimum register blocking is 2x1 (SIMDizable)
- Can increase memory traffic by 66%

Cache blocking optimization is transformed into local store blocking
- Spatial and temporal locality is captured by software when the matrix is optimized
  - In essence, the high bits of column indices are grouped into DMA lists

No branch prediction
- Replace branches with conditional operations

In some cases, what were optional optimizations on cache based machines, are requirements for correctness on Cell

Despite the performance, Cell is still handicapped by double precision
Performance

Intel Clovertown

AMD Opteron

Sun Niagara2

IBM Cell Broadband Engine

39% of peak flops
89% of bandwidth
Multicore MPI Implementation

- This is the default approach to programming multicore
- Used PETSc with shared memory MPICH
- Used OSKI (developed @ UCB) to optimize each thread
- = Highly optimized MPI
Summary
Median Performance & Efficiency

- Used digital power meter to measure sustained system power
  - FBDIMM drives up Clovertown and Niagara2 power
  - Right: sustained MFlop/s / sustained Watts
- Default approach (MPI) achieves very low performance and efficiency
Paradoxically, the most complex/advanced architectures required the most tuning, and delivered the lowest performance.

Most machines achieved less than 50-60% of DRAM bandwidth

Niagara2 delivered both very good performance and productivity

Cell delivered very good performance and efficiency
   - 90% of memory bandwidth
   - High power efficiency
   - Easily understood performance
   - Extra traffic = lower performance (future work can address this)

multicore specific autotuned implementation significantly outperformed a state of the art MPI implementation

✓ Matrix compression geared towards multicore
✓ NUMA
✓ Prefetching
Acknowledgments

- UC Berkeley
  - RADLab Cluster (Opterons)
  - PSI cluster (Clovertowns)
- Sun Microsystems
  - Niagara2
- Forschungszentrum Jülich
  - Cell blade cluster
Questions?