Optimizing Sparse Matrix-Vector Operations on Scalar and Vector Processors

Dinesh Kaushik

William Gropp

Argonne National Laboratory

Organization of the Presentation

- Factors limiting performance
- Performance analysis of sparse matrix-vector multiplication on Scalar Processors
- Similar analysis for Cray X1
- Conclusions and Future Work

Performance Modeling and Prediction

- Important to know what is “achievable” performance
  - Peak performance – very loose upper bound
  - Benchmarks based on Dense Linear Algebra – does not represent PDE workload
- What to expect from a new architecture
  - If memory bandwidth gets doubled, which computational phase benefits the most?

Three Fundamental Limiting Factors to Peak Performance

- Memory Bandwidth
  - Processor does not get data at the rate it requires
- Instruction Issue Rate
  - If the loops are load/store bound, we will not be able to do a floating point operation in every cycle even if the operands are available in primary cache
  - Several constraints (like primary cache latency, latency of floating point units etc.) are to be observed while coming up with an optimal schedule
- Fraction of Floating Point Operations
  - Not every instruction is a floating point instruction

Analyzing A Simple Kernel: Sparse Matrix Vector Product

- Sparse matrix vector product is important part of many iterative solvers
- Its performance modeling is straightforward
- We present simple analysis to predict better performance bounds (based on the three architectural limits) than the “marketing” peak of a processor

Performance Issues for Sparse Matrix Vector Product

- Little data reuse
- High ratio of load/store to instructions/floating-point ops
- Stalling of multiple load/store functional units on the same cache line
- Low available memory bandwidth

Sparse Matrix Vector Algorithm: A General Form

for every row, i {

  fetch ia(i+1)

  for j = ia(i) to ia(i + 1) {   // loop over the non-zeros of the row

    fetch ja(j), a(j), x_1(ja(j)), ...x_N(ja(j))

    do N fmadd (floating multiply add)

  }

  Store y_1(i) .....y_N(i)

}

Estimating the Memory Bandwidth Limitation

Assumptions

- Perfect Cache (only compulsory misses; no overhead)
- No memory latency
- Unlimited number of loads and stores per cycle

Data Volume (AIJ Format)

m*sizeof(int) + N*(m+n)*sizeof(double)

// ia, N input (size n) and output (size m) vectors

+ Nnz* (sizeof(int) + sizeof(double))

// ja, and a arrays

= 4*(m+nnz) + 8*(N*(m+n)+ Nnz)

Estimating the Memory Bandwidth Limitation (Contd.)

- Number of Floating-Point Multiply Add (fmadd) Ops = N*nz
- For square matrices,

$$\text{Bytes transferred/fmadd} = (16 + \frac{4}{N})*\frac{n}{N_{nz}} + \frac{12}{N}$$

(Since Nnz >> n, Bytes transferred / fmadd ~12/N)

- Similarly, for **Block AIJ** (**BAIJ**) format

$$\text{Bytes transferred/fmadd} = (16 + \frac{4}{N*b})*\frac{n}{N_{nz}} + (\frac{4}{N*b} + \frac{8}{N})$$

Performance Summary on 2.4 GHz P4 Xeon

- Matrix size, n = 90,708; number of nonzero entries, Nnz = 5,047,120 (from computational aerodynamics, b=4)
- Stream performance is 1973 MB/sec (for triad vector operation) http://www.cs.virginia.edu/stream
- Number of Vectors, N = 1, and 4

Format	Number of Vectors	Bytes / flop	Bandwidth (GB/s)		MFlops	
			Required	Achieved	Ideal	Achieved
AIJ	1	6.18	14.83	1.97	319	274
AIJ	4	1.66	3.98	1.97	1188	610

Estimating the Operation Issue Limitation

AT:address transln; **Br**: branch; **Iop**: integer op; **Fop**: floating point op; **Of**: offset calculation; **Ld**: load; **St**: store

```
for (i = 0, i < m; i++) {
        jrow = ia(i+1)                                          // 1Of, AT, Ld
        ncol =  ia(i+1) -ia(i)                                  // 1 Iop
        Initialize, sum_1 .....sum_N                            //  N Ld
        for (j = 0; j < ncol; j++) {                            // 1 Ld
          fetch ja(jrow), a(jrow), x_1(ja(jrow)), .....x_N(ja(jrow))
                                                                // 1 Of, N+2 AT, and  Ld
          do N fmadd (floating multiply add)                    // 2N Flop
        }                                                       // 1 Iop, 1 Br
        Store sum_1.....sum_N in y_1(i) .....y_N(i)             // 1 Of, N AT, and St
}                                                               // 1 Iop, 1 Br
```

Estimating the Operation Issue Limitation (Contd.)

- Assumptions:
  - Data items are in cache
  - Each operation takes only one cycle to complete but multiple operation can graduate in one cycle
- If only one load or store can be issued in one cycle, the best we can hope for is

$$\frac{\text{Number of floating point instructions}}{\text{Number of Loads and Stores}} * \text{Peak MFlops/s}$$

- Other restrictions (like primary cache latency, latency of floating point units etc.) need to be taken into account while creating the best schedule (especially on those processors where software pipelining is important)

Estimating the Fraction of Floating Point Operations

- Assumptions:
  - infinite number of functional units
  - data items are in primary cache
- Estimated number of floating point operations out of the total instructions:

$$\text{Total number of instructions completed } (I_t) = m*(3*N+8) + N_{nz}*(4*N+9)$$

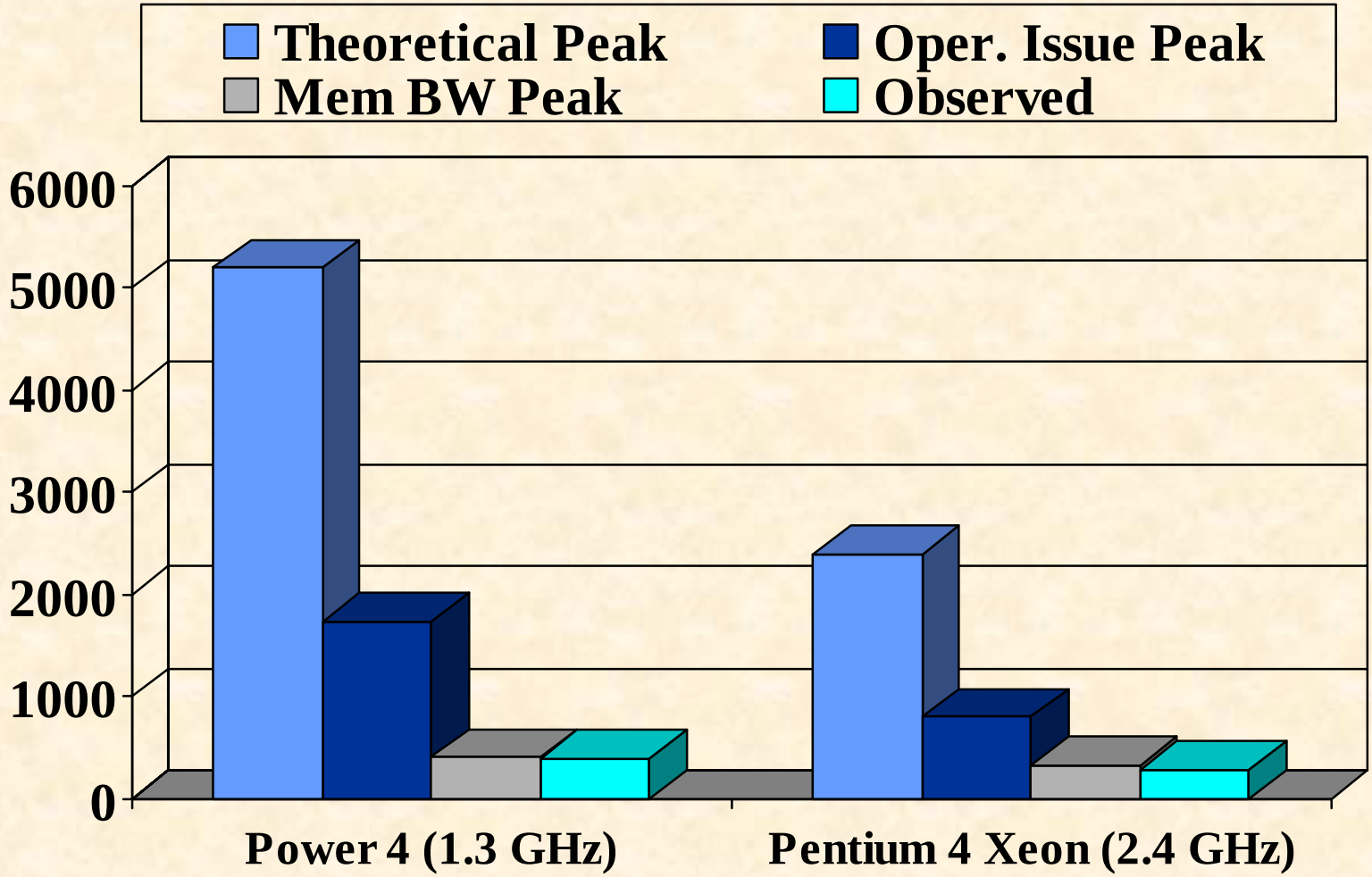
$$\text{Fraction spent on floating point work } (I_f) = \frac{2*N*N_{nz}}{m*(3*N+8) + N_{nz}*(4*N+9)}$$

- For **N=1, I_f = 0.18 and N = 4, I_f = 0.34.**

Realistic Measures of Peak Performance

Sparse Matrix Vector Product

One vector, matrix size, m = 90,708, nonzero entries nz = 5,047,120



Cray X1 Architecture

- Basic building block is multistreaming processor (MSP) having four single streaming processors (SSPs)
- Each SSP runs at 400 MHz for scalar units and 800 MHz for vector unis
- Each MSP can do 16 vector adds and 16 vector mutliply in one cycle, peaking at 12.8 Gflops/s for 800 MHz vector units

Cray X1 Memory Hierarchy

- Four SSPs on a MSP share a 2 MB cache
- Four MSPs share 16GB of memory forming a Cray X1 node
- Ideal load bandwidth from memory to cache for an MSP is 8 double words per clock cycle (~25.6 GB/s for 400 MHz clock)
- The ideal load bandwidth (BWe) from cache to processor is 16 double words per cycle (~51.2 GB/s)

CSR format on Cray X1

- CSR format can only vectorize on per-row basis
- Gives poor performance since the number of non-zeroes in each row is small
- We get only about 50 Mflop/s with one vector and 130 Mflops/s with four vectors for the PETSc-FUN3D matrix described earlier

Other Formats for Vector Processors

- Sparse diagonal – useful for PDEs
- Ellpack/Itpack (ELL)
  - forces all rows to have the same length as the longest row by padding zeros
  - Vectorization is done on columns of the modified matrix
- Jagged Diagonal (JAD)
  - Permute the matrix in the order of decreasing length
  - First jagged diagonal is constructed by extracting the first element from each row, second by extracting the second element, and so on

Analyzing the Diagonal Format

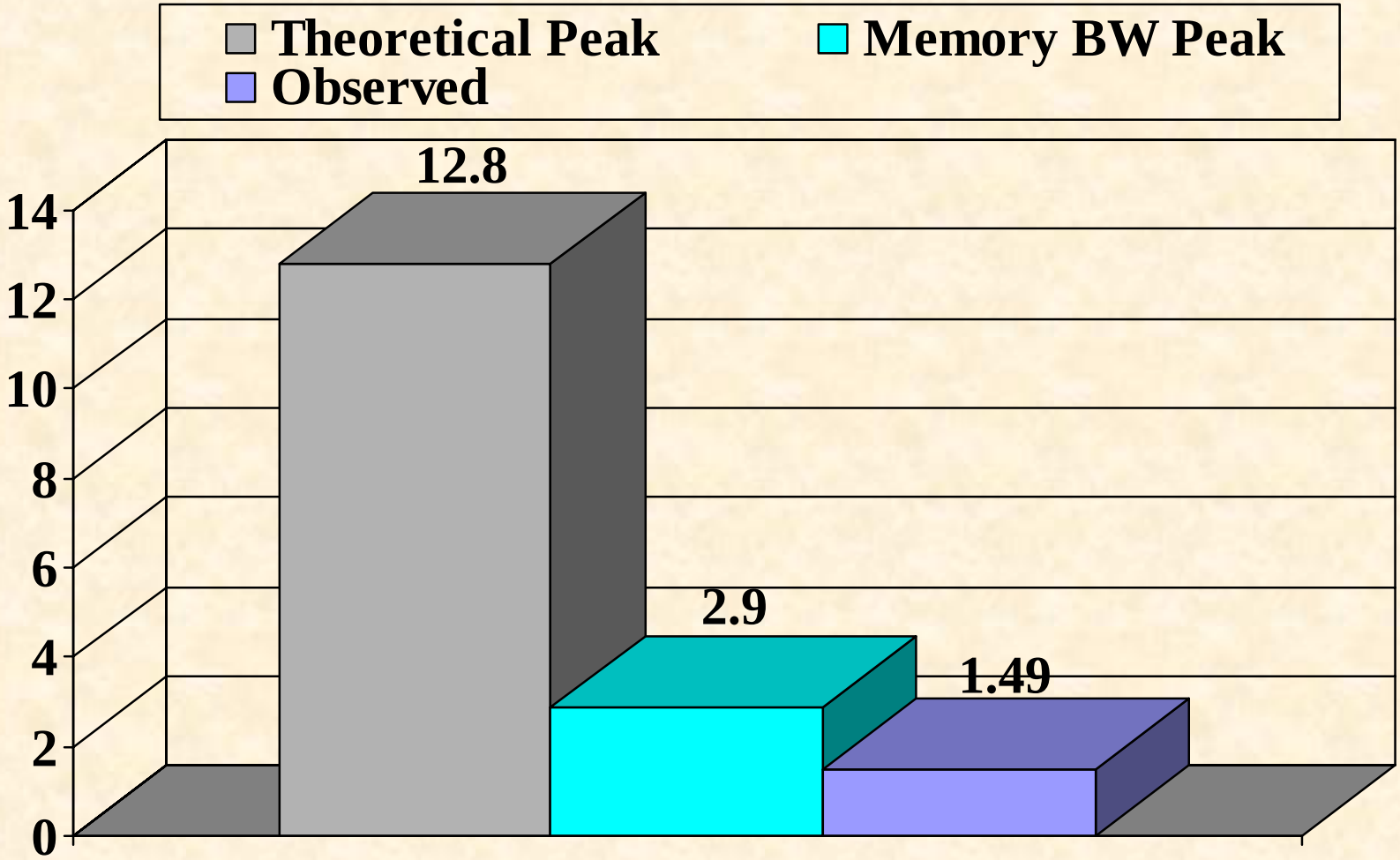
- Effective bandwidth:

  $BW_{eff}(H) = \min[BW_c, BW_m/(1-H)]$,

  where H is cache hit rate
- Achieved memory bandwidth is about 70% of BW_m
- y[j+col] += a[j]*x[j+col]
- Assume that the input and output vectors are in cache; therefore, we need to bring only the diagonal and column index arrays from main memory
- In this ideal situation, we bring 12 bytes for every 2 flops from main memory; bytes/flops = 6, leading to only **23 % of** machine peak**.**

Sparse Matrix vector Product on Cray X1

One vector, matrix size, m = 90,708, nonzero entries nz = 5,047,120



Conclusions and Future Work

- Using multivectors can improve the performance of sparse matrix-vector product significantly
- Simple models predict the performance of sparse matrix-vector operations on a variety of platforms, including the effects of memory bandwidth, and instruction issue rates
  - achievable performance is a small fraction of stated peak for sparse matrix-vector kernels, independent of code quality
- Even though memory bandwidth is huge on Cray X1 (as compared to most scalar machines), sparse matvec is still memory bandwidth limited; only 23 % of machine peak is possible under ideal situations.
- In future, we plan to study Jagged diagonal and Segmented scan formats

Acknowledgements

- James Schwarzmeier of Cray Research Inc., Satish Balay and Barry Smith of Argonne, and David Keyes of Columbia University for many interesting discussions
- Oak Ridge National Laboratory for computer time on IBM power 4 and Cray X1
- Argonne National Laboratory for computer time on Jazz cluster