Parallel Architecture

Announcements
- HW0 is due Friday night, thank you for those who have already submitted
- HW1 is due Wednesday night

Today
- Computing operational intensity
- Dwarves and Motifs
- Stencil computation demo
- Touchstone apps for 560 Spring 2012
  - Dynamic programming: Protein string matching
  - Sparse Linear Algebra: SpMV
  - Structured grids: stencil computations with implicit and explicit coefficients
  - N-body methods: n-body stars (HW3) and molecular dynamics
  - Dense Linear Algebra: Matrix matrix multiply, forward and backward substitution, and Cholesky (later)

Roofline Model (Will be using this in HW2)

Roofline: An Insightful Visual Performance Model for Multicore Architecture
- By Sam Williams, Andrew Waterman, and David Patterson

Operational Intensity: Operations per byte of DRAM traffic.

Roofline graph per machine
- FLOPS/sec versus operational intensity
- Horizontal line for the peak floating point performance (compute bound)
- Diagonal line for the measured peak memory performance (memory bound)

Placing ceilings to represent how performance optimizations can help
- Improve ILP and apply SIMD (computation bound)
- Balance floating point operation mix (computation bound)
- Unit stride accesses (memory bound)
- Memory affinity (memory bound)
Operational Intensity for SpMV

Sparse Matrix Vector Product \( y = Ax \)

Operational Intensity is flops/(byte of memory traffic)

Computed using coordinate storage (COO)

```
for (i=0; i<N; i++) { Y[i] = 0; }
for (p=0; p<NNZ; p++) {
    Y[row[p]] += val[p]*X[col[p]];
}
```

Computed using more common compressed sparse row (CSR)

<demo CSR>

```
for (i=0; i<N; i++) {
    y = 0;
    for (p=rowptr[i]; p<rowptr[i+1]; p++) {
        y += val[p]*X[col[p]];
    }
    Y[i] = y;
}
```

The Berkeley View

Conventional Wisdom and their replacements <read and discuss in class>

13 Dwarfs/Motifs

- Dense linear algebra, dense matrices and vectors, matrix-matrix, matrix vector
- Sparse linear algebra, explicitly store only non-zeros, explicit storage of indices
- Spectral methods, FFT, specific pattern of data permutation, all to all communication
- N-body methods, interactions between discreet points, various algorithms
- Structured grids, regular grid, neighbor relationships implicit in multi-dim array structure
- Unstructured grids, irregular grid, connectivity of grid must be explicit
- Monte carlo, repeated random trials with some final summary, map-reduce
- Combinational Logic, logical functions with stored state
- Graph traversal, e.g. quicksort
- Dynamic programming, filling a table with solutions to subproblems to build final solution
- Backtrack and Branch and Bound, recursive division of feasible solution space with pruning
- Construct Graphs, e.g., Hidden Markov Models and Bayesian networks
- Finite State Machine, serial with single state at one time and transitions to other states
1D Stencil Computation

Stencil Computations
- Computations operate over some mesh or grid
- Computation is modifying the value of something over time or as part of a relaxation to find steady state
- Each computation has some nearest neighbor data dependence pattern
- The coefficients multiplied by neighbor can be constant or variable

1D Stencil Computation version 1 <demo in class>
// assume A[0,i] initialized to some values
for (t=1; t<(T+1); t++) {
    for (i=1; i<(N-1); i++) {
    }
}

1D Stencil Computation (take 2)

1D Stencil Computation, version 2 <demo in class>
// assume A[i] initialized to some values
for (t=0; t<T; t++) {
    for (i=1; i<(N-1); i++) {
    }
}

Analysis
- Are version 1 and version 2 computing the same thing?
- What is the operational intensity of version 1 versus version 2?
- What parallelism is there in version 1 versus version 2?
Jacobi in SWM code (Stencil Computation with Explicit Weights)

Source: David Randall’s research group

Given an $N \times N$ lower triangular matrix with unit diagonals and a $n$-vector $b$ solve for the vector $x$ in $Lx = b$

$$b_i = \sum_{j=1}^{N} L_{i,j} x_j$$

How do we solve for $x$?

How do we turn this into a loop program?
Moldyn <draw iteration space>

```c
for (tstep=0; tstep<=n_tstep-1; tstep++) {
    ...
    for (i=0; i<=n_moles-1; i++) {
        x(i) = x(i) + vhx(i) + fx(i);
        ...
        if ( x(i) < 0.0 ) x(i) = x(i) + side;
        if ( x(i) > side ) x(i) = x(i) - side;

        vhx(i) = vhx(i) + fx(i);
        fx(i) = 0.0;
    }
    for (ii=0; ii<=n_inter-1; ii++) {
        i = inter1(ii);  j = inter2(ii);
        fx(i) += ... x(i)... x(j)... 
        fx(j) += ... x(i)... x(j)...
    }
    for (i=0; i<=n_moles-1; i++) {
        ...
        vhx(i) = ... fx(i) ...;
    }
}
```

Concepts

**Computing operational intensity**

**Berkeley dwarves/motifs**
- What they are and examples
- Their parallel performance properties
- Explicit versus implicit storage of indices, graph connectivity, etc.

**Stencil computations**
- Nearest neighbor data dependences

**Touchstone apps for the class**
- The Berkeley dwarf/motif categories they represent.
- Data reuse within the touchstone apps
- Parallelism within the touchstone apps
Next Time

Reading
- Advanced Compiler Optimizations for Supercomputers by Padua and Wolfe

Homework
- HW0 is due Friday 1/27/12
- HW1 is due Wednesday 2/1/12

Lecture
- Parallelization and Performance Optimization of Applications
COO coordinate storage

\[
\begin{array}{c|c|c|c|c|c}
\text{row} & 0 & 1 & 2 & 3 & 4 \\
\hline
\text{col} & 0 & 1 & 2 & 3 & 4 \\
\hline
\text{val} & \ast & \ast & \ast & \ast & \ast \\
\end{array}
\]

\[\text{n}_{\text{nz}} < 10\% (\text{n}_{\text{row}} \times \text{n}_{\text{col}})\]

0 dense in 2D array

CSR compressed sparse row

\[
\begin{array}{c|c|c|c|c|c|c|c}
\text{row} & 0 & 1 & 2 & 3 & 4 & 5 & \text{NN}_1 \text{NN}_1 \\
\hline
\text{col} & 2 & 1 & 2 & 3 & 4 & 0 & 2 & 4 & 1 \\
\hline
\text{val} & A & B & C & D & E & F & G & H & I \\
\end{array}
\]

\[\text{n}_{\text{nz}} \]

\[\text{N} = \text{n}_{\text{row}} \times \text{n}_{\text{col}}\]
for (p = 0 to nnz - 1

\[
\begin{align*}
  y_{\text{row}[p]} &= y_{\text{row}[p]} + x_{\text{col}[p]} \times \text{val}[p] \\
  y_{\text{col}} &= \text{col}
\end{align*}
\]

2 flops
loads: row[p], col[p], y[...], val[...], x[...]

32
assumptions: i in reg, row[p] loaded once
to all others go to memory

\[
\text{operational intensity} = \frac{2}{32} = 0.0625
\]

\[
\text{arithmetic intensity} = \frac{0.17}{0.25} ?
\]

assumption: everything brought into memory once

bytes = nnz*(16) + 2N(8)
flops = 2\nnz
\[
\text{operational intensity} = \frac{2 \times \text{nnz}}{16 \times \text{nnz} + 16 \times N}
\]

.125