CS4961 Parallel Programming

Lecture 12: Data Locality, cont.

Mary Hall
September 30, 2010
Combining Locality, Thread and SIMD Parallelism:

The following code excerpt is representative of a common signal processing technique called convolution. Convolution combines two signals to form a third signal. In this example, we slide a small (32x32) signal around in a larger (4128x4128) signal to look for regions where the two signals have the most overlap.

```c
for (l=0; l<N; l++) {
    for (k=0; k<N; k++) {
        C[k][l] = 0.0;
        for (j=0; j<W; j++) {
            for (i=0; i<W; i++) {
                C[k][l] += A[k+i][l+j]*B[i][j];
            }
        }
    }
}
```
Programming Assignment 2, cont.

• Your goal is to take the code as written, and by either changing the code or changing the way you invoke the compiler, improve its performance.

• You will need to use an Intel platform that has the “icc” compiler installed.
  - You can use the CADE Windows lab, or another Intel platform with the appropriate software.
  - The Intel compiler is available for free on Linux platforms for non-commercial use.
  - The version of the compiler, the specific architecture and the flags you give to the compiler will drastically impact performance.

• You can discuss strategies with other classmates, but do not copy code. Also, do not copy solutions from the web. This must be your own work.
Programming Assignment 2, cont.

• How to compile
  - OpenMP (all versions): icc -openmp conv-assign.c

• Measure and report performance for five versions of the code, and turn in all variants:
  - Baseline: compile and run code as provided (5 points)
  - Thread parallelism only (using OpenMP): icc -openmp conv-omp.c (5 points)
  - SSE-3 only: icc -openmp -msse3 -vec-report=3 conv-sse.c (10 points)
  - Locality only: icc -openmp conv-loc.c (10 points)
  - Combined: icc -openmp -msse3 -vec-report=3 conv-all.c (15 points)

• Explain results and observations (15 points)

• Extra credit (10 points): improve results by optimizing for register reuse
Hints and Suggestions on the Assignment

• There are no absolute answers
  - And the only incorrect answers are when you change the program and get the wrong answer

• The goal is to improve performance through optimization

• You can only speculate on what is happening based on measured performance
  - Observe changes in performance
  - You may even be wrong, but the important thing is to have a reasoned argument about why the performance changed
Reordering Transformations, cont.

• Analyze reuse in computation

• Apply loop reordering transformations to improve locality based on reuse

• With any loop reordering transformation, always ask
  - Safety? (doesn’t reverse dependences)
  - Profitability? (improves locality)
Loop Permutation:
An Example of a Reordering Transformation

Permute the order of the loops to modify the traversal order

```
for (i=0; i<3; i++)
for (j=0; j<6; j++)
```

```
for (j=0; j<6; j++)
for (i=0; i<3; i++)
```

Which one is better for row-major storage?
Permutation has many goals

• Locality optimization
  • Particularly, for spatial locality (like in your SIMD assignment)

• Rearrange loop nest to move parallelism to appropriate level of granularity
  - Inward to exploit fine-grain parallelism (like in your SIMD assignment)
  - Outward to exploit coarse-grain parallelism

• Also, to enable other optimizations
Tiling (Blocking): Another Loop Reordering Transformation

- Blocking reorders loop iterations to bring iterations that reuse data closer in time
- Goal is to retain in cache between reuse
Tiling is Fundamental!

• Tiling is very commonly used to manage limited storage
  - Registers
  - Caches
  - Software-managed buffers
  - Small main memory

• Can be applied hierarchically
  - You can tile a tile
Tiling Example

```c
for (j=1; j<M; j++)
    for (i=1; i<N; i++)
        D[i] = D[i] + B[j,i]
```

**Strip mine**

```c
for (j=1; j<M; j++)
    for (i=1; i<N; i+=s)
        for (ii=i; ii<min(i+s-1,N); ii++)
            D[ii] = D[ii] + B[j,ii]
```

**Permute**

```c
for (i=1; i<N; i+=s)
    for (j=1; j<M; j++)
        for (ii=i; ii<min(i+s-1,N); ii++)
            D[ii] = D[ii] + B[j,ii]
```
How to Determine Safety and Profitability?

• Safety
  - A step back to Lecture 2
  - Notion of reordering transformations
  - Based on data dependences
  - **Intuition:** Cannot permute two loops i and j in a loop nest if doing so reverses the direction of any dependence.
    - Tiling safety test is similar.

• Profitability
  - Reuse analysis (and other cost models)
  - Also based on data dependences, but simpler
Simple Permutation Examples: 2-d Loop Nests

\[
\begin{align*}
\text{for } (i = 0; i < 3; i++) \\
\quad &\text{for } (j=0; j < 6; j++) \\
\end{align*}
\]

\[
\begin{align*}
\text{for } (i = 0; i < 3; i++) \\
\quad &\text{for } (j=1; j < 6; j++) \\
\end{align*}
\]

• Ok to permute?
Legality of Tiling

- Tiling = strip-mine and permutation
  - Strip-mine does not reorder iterations
  - Permutation must be legal
  OR
  - strip size less than dependence distance
Unroll-and-jam

Unroll outer loops and fuse ("jam") copies of inner loop together
Can be achieved with tiling and unrolling (next examples)
Assume M is divisible by 2

```c
for (i=0; i<M; i++)
    for (j=0; j<N; j++)
```

```c
for (i=0; i<M; i+=2)
    for (j=0; j<N; j++)
    for (j=0; j<N; j++)
```

```c
for (i=0; i<M; i+=2)
    for (j=0; j<N; j++)
```
Locality + SIMD (SSE-3) Example

Example: matrix multiply

for (J=0; J<N; J++)
    for (K=0; K<N; K++)
        for (I=0; I<N; I++)
Locality + SIMD (SSE-3) Example

Tiling inner loops I and K (+permutation)

for (K = 0; K<N; K+=T_K)
  for (I = 0; I<N; I+=T_I)
    for (J =0; J<N; J++)
      for (KK = K; KK<min(K+T_K, N); KK++)
        for (II = I; II<min(I+ T_I, N); II++)
Locality + SIMD (SSE-3) Example

Unroll II loop, $T_I = 4$ (equiv. to unroll\&jam)

```c
for (K = 0; K<N; K+=T_K)
    for (I = 0; I<N; I+=4)
        for (J =0; J<N; J++)
            for (KK = K; KK<min(K+T_K, N); KK++)
```

Now parallel computations are exposed
Scalar Replacement: Replace accesses to C with scalars

for (K = 0; K<N; K+=T_K)
   for (I = 0; I<N; I+=4)
      for (J =0; J<N; J++) {
         C0 = C[J][I]; C1 = C[J][I+1]; C2 = C[J][I+2], J; C3 = C[J][I+3];
         for (KK = K; KK<min(K+T_K, N); KK++) {
            C0 = C0 + A[KK][II] * B[J][KK];
            C1 = C1 + A[KK][II+1] * B[J][KK];
            C2 = C2 + A[KK][II+2] * B[J][KK];
            C3 = C3 + A[KK][II+3] * B[J][KK];
         }
         C[J][I] = C0; C[J][I+1] = C1; C[J][I+2] = C2; C[J][I+3] = C3;
      }

Now C accesses can be mapped to “named registers”
Locality + SIMD (SSE-3) Example

Different direction: SIMD operations can be performed in SSE-3
- Array notation and "broadcast" implies SSE-3 operations
- Compiler will take care of this if the code is in the right form

```
for (K = 0; K<N; K+=T_K)
  for (I = 0; I<N; I+=4)
    for (J =0; J<N; J++)
      for (KK = K; KK<min(K+T_K, N); KK++)
        Btemp[0:3] = broadcast of B[J][KK] to all fields
```
Locality + SIMD (SSE-3) Example

Now unroll K loop, $T_k = 4$ (equiv. to unroll & jam)

```
for (K = 0; K<N; K+=4)
  for (I = 0; I<N; I+=4)
    for (J =0; J<N; J++) {
      Btemp[0:3] = broadcast of B[J][K] to all fields
      Btemp[0:3] = broadcast of B[J][K+1] to all fields
      Ctemp[0:3] = Ctemp[0:3] + A[K+1][I:I+3] * Btemp[0:3]
      Btemp[0:3] = broadcast of B[J][K+2] to all fields
      Ctemp[0:3] = Ctemp[0:3] + A[K+2][I:I+3] * Btemp[0:3]
      Btemp[0:3] = broadcast of B[J][K+3] to all fields
      Ctemp[0:3] = Ctemp[0:3] + A[K+3][I:I+3] * Btemp[0:3]
    }
```

“Superword” replacement
Uses SSE-3 register file