CS4961 Parallel Programming

Lecture 15: Locality/VTUNE Homework

Mary Hall
October 27, 2009

Administrative

- Homework assignment 3 will be posted today (after class)
- Due, Thursday, November 5 before class
  - Use the "handin" program on the CADE machines
  - Use the following command:

    "handin cs4961 hw3 <gzipped tar file>"
- Mailing list set up: cs4961@list.eng.utah.edu
- Next week we'll start discussing final project
  - Optional CUDA or MPI programming assignment part of this

Midterm Exam

Score Range	Number of Students	Grade
97-100	1	A+
88-93	6	A
85-87	4	A-
80-83	7	B+
75-79	7	B
72-73	2	B-
59-61	2	C

Comments on Exam

- Overall, most students did well with the concepts
- Some problem-solving gaps
- Quick discussion of questions

Exam discussion

Problem 2:

a. A multiprocessor consists of 100 processors, each capable of a peak execution rate of 2 Gflops (i.e., 2 billion floating point operations per second). What is the peak performance of the system as measured in Gflops for an application where 10% of the code is sequential and 90% is parallelizable?

Key point: Speedup roughly 10, so roughly 20 GFlops

Exam discussion

b. Given the following code, which is representative of a Fast Fourier Transform:

```
procedure FFT_like_pattern(A,n) {
float *A;
int n, m;

m = log2n;
for (j=0; j<m; j++) {
   k = 2^j;
   for (i=0; i<k; i++)
      A[i] = A[i] + A[i XOR 2^j];
}
}
```

Key points: main dependence on j loop, parallelize I loop

(1) What are the data dependences on loops i and j?
(2) Assume n = 16. Provide OpenMP or Peril-L code for the mapping to a shared-memory parallel architecture.

Exam discussion

(c) Construct a task-parallel (similar to producer-consumer) pipelined code to identify the set of prime numbers in the sequence of integers from 1 to n. A common sequential solution to this problem is the sieve of Erasthones. In this method, a series of all integers is generated starting from 2. The first number, 2, is prime and kept. All multiples of 2 are deleted because they cannot be prime. This process is repeated with each remaining number, up until but not beyond sqrt(n). A possible sequential implementation of this solution is as follows:

```
for (i=2; i<=n; i++) {
   prime[i] = true;
for (i=2; i<= sqrt(n); i++) {
   if (prime[i]) {
      for (j=i+i; j<=n; j = j+i) { // multiples of i are set to non-prime
         prime[j] = false;
   }
}
```

Key points: Task parallelism, buffer for queuing data so no data dependences, modify indexiing

Homework 3

Problem 1. Assume a cache line size of 4 elements. Identify the different kinds of reuse and how many memory accesses there are in the following example, assuming (a) row-major order, (b) column-major order. Use the inner loop memory cost calculation from slides 11-13 of Lecture 15 to estimate memory accesses.

```
for (i = 0; i<n; i++)
  for (j = 0; j<m; j++)
    A[i][j] = B[i][j] + B[j][i] + C[i]
```

Homework 3, cont.

Problem 2. What code would be generated to tile the following loop nest for reuse in cache, assuming row-major order and two levels of tiling (Note: the loop order may need to be modified). Prove that tiling is safe.

```
for (i = 0; i<n; i++)
   for (j = 0; j<m; j++)
     for (k=0; k<l;k++)
        A[i][k] = A[i][k] + B[k][j]*C[k][k]
```


Homework 3

Problem 3: VTUNE:

Consider the jacobi code in jacobi.c on the website. Here is the main computation:

```
// play around with this loop nest
  for (i=1; i<width-1; i++)
   for (j=1; j<height-1; j++)
     A[i][j] = (B[i+1][j] + B[i-1][j] + B[i][j+1] + B[i][j-1])/4;
```

(a) Run this code under VTUNE. Indicate event-based sampling, and collect the following events. (CPU_CLK_UNHALTED, MEM_LOAD_RETIRED.L1D_MISS, MEM_LOAD_RETIRED.L2_MISS)

(b) Now attempt to tile the innermost loop and repeat. Do you see an impact on cache misses and cycles.

(c) Extra credit: Tile the other loop. Now what happens.

Reuse Analysis: Use to Estimate Cache Misses

Remember: Row-major storage for C arrays

```
for (i=0; i<N; i++)
   for (j=0; j<M; j++)
      A[i]=A[i]+B[j][i]
```

```
for (j=0; j<M; j++)
   for (i=0; i<N; i++)
      A[i]=A[i]+B[j][i]
```

reference	*loop* J	*loop* I
A[i]	1	N
B[j,i]	M	N*M

reference	*loop* I	*loop* J
A[i]	N/cls(*)	M*N/cls
B[j,i]	N/cls	M*N/cls

(*) cls = Cache Line Size (in elements)

Allen & Kennedy: Innermost memory cost

- Innermost memory cost: $C_M(L_i)$
  - assume L_i is innermost loop
    - l_i = loop variable, N = number of iterations of L_i
  - for each array reference **r** in loop nest:
    - **r** does not depend on l_i : cost (**r**) = 1
    - **r** such that l_i strides over a non-contiguous dimension: cost (**r**) = N
    - **r** such that l_i strides over a contiguous dimension: cost (**r**) = N/cls
  - At outer loops,
    - multiply cost(r) by trip count if reference varies with loop index
    - Otherwise, multiply cost(r) by 1 unless pushed out of cache
  - $C_M(L_i)$ = sum of cost (**r**)

Implicit in this cost function is that N is sufficiently large that cache capacity is exceeded by data footprint in innermost loop

Canonical Example: Selecting Loop Order

```
for (i=0; i<N; i++)
   for (j=0; j<N; j++)
         C[i][j] = 0;
         for (k=0; k<N; k++)
          C[i][j]= C[i][j] + A[i][k] * B[k][j];
```

- C_M(i) = $2N^3 + N^2$ [C: N^3, A: N^3, B: N^2)
- C_M(j) = $2N^3$/cls + N^2 [C: N^3/cls, A: N^2, B: N^3/cls]
- C_M(k) = N^2 + N^3/cls + N^3 [C: N^2, A: N^3/cls, B: N^3]
- Ordering by innermost loop cost: j,k,i

Canonical Example: Selecting Tile Size

Choose T_i and T_k such that data footprint does not exceed cache capacity

```
DO K = 1, N  by TK
   DO I = 1, N by TI
    DO J = 1, N
       DO KK = K, min(KK+ TK,N)
        DO II = I, min(II+ TI,N)
           C(II,J)= C(II,J)+A(II,KK)*B(KK,J)
```