Shared-memory Parallel Programming with Cilk Plus

Milind Chabbi

Nikhil Hegde

Department of Computer Science
IIT Dharwad

CS410 Lecture 8 19 January 2024

Outline for Today

- **Cilk Plus (Cont…)**
  - **—task parallelism examples**
    - **cilksort**
  - **—explore speedup and granularity**
  - **—parallel loops**
  - **—reducers**

Cilksort

Variant of merge sort

```
void cilksort(ELM *low, ELM *tmp, long size) {
    long quarter = size / 4;
    ELM *A, *B, *C, *D, *tmpA, *tmpB, *tmpC, *tmpD;
    if (size < QUICKSIZE) { seqquick(low, low + size - 1) return; }

    A = low; tmpA = tmp;
    B = A + quarter; tmpB = tmpA + quarter;
    C = B + quarter; tmpC = tmpB + quarter;
    D = C + quarter; tmpD = tmpC + quarter;

    cilk_spawn cilksort(A, tmpA, quarter);
    cilk_spawn cilksort(B, tmpB, quarter);
    cilk_spawn cilksort(C, tmpC, quarter);
    cilksort(D, tmpD, size - 3 * quarter);
    cilk_sync;

    cilk_spawn cilkmerge(A, A + quarter - 1, B, B + quarter - 1, tmpA);
    cilkmerge(C, C + quarter - 1, D, low + size - 1, tmpC);
    cilk_sync;

    cilkmerge(tmpA, tmpC - 1, tmpC, tmpA + size - 1, A);
}
```

Merging in Parallel

- **How can you incorporate parallelism into a merge operation?**
- **Assume we are merging two sorted sequences A and B**
- **Without loss of generality, assume A larger than B**

Algorithm Sketch

1. **Find median of the elements in A and B (considered together).**
2. **Do binary search in A and B to find its position. Split A and B at this place to form A_1, A_2, B_1, and B_2**
3. **In parallel, recursively merge A_1 with B_1 and A_2 with B_2**

See https://www.geeksforgeeks.org/median-two-sorted-arrays-different-sizes-ologminn-m for computing the median of two sorted sequences in O(log(min(n,m)) time, where $|A| = n$ and $|B| = m$

Optimizing Performance of cilksort

- Recursively subdividing all the way to singletons is <u>expensive</u>
- When the size of the remaining sequence to sort or merge is small (e.g., 2K)
  - use sequential quicksort
  - use sequential merge

Speedup Demo

Explore speedup of naive fibonacci program

- ssh paramutkarsh.cdac.in -p 4422 cluster
- fib.cpp: a Cilk program for computing nth fibonacci number
- Compile: `clang++ -O2 -g -fopencilk -std=c++11 fibcpp -o fib.cilk`
- experiment with the fibonacci program
- `bash cilk_run.sh fib 47 |& tee out.txt`
  - *computes fib(47) with 1, 2, 3, 4, 8, 16, and 32*
  - *CILK_NWORKERS on the master node (which has 80 hardware threads)*
- what value of CILK_NWORKERS yields the lowest execution time?
- what is the speedup vs. the execution time of "`./fib-serial 47`"?
- how does this speedup compare to the total number of HW threads?

Granularity Demo

Explore how increasing the granularity of parallel work in fib improves performance (by reducing c_1)

- fib_cutoff.cpp: a program for computing n^{th} fibonacci #
  - this version differs in that after going H levels deep, it stops spawning parallel work all the way down
- Experiment with the fibonacci program with reduced parallelism
  - compute fib(47) for H = 10
- What is the lowest execution time?
- What is the speedup vs. the execution time of
  - "`./fib-serial 47`"?