Parallel, In-order Creation of lists

Pablo Halpern, 2 May 2012

In Kona, I presented N3361, which advocates the position that C++ should have language constructs for parallel programming. The presentation included an overview of Intel® Cilk Plus™, including a unique library feature called *reducers*. Unfortunately, my original example for reducers did not illustrate one of the primary advantages of reducers over other parallel reduction constructs such as the **Combinable** template in Microsoft's PPL*: that the reduction operation needs to be associative but not necessarily commutative. This means, for example, that you can use a reducer to create linked list because the list-append operation is associative. The resulting linked list will have the same order of elements as the same program with the Cilk keywords elided (i.e., a serial program that looks like the Cilk program but runs with only one thread).

I have heard some skepticism that our list reducer could produce a list in the same order as the serial program without paying a performance penalty. To prove the point, I created a benchmark program called reducer_proof.cpp, which is attached to the SG1 Wiki page. (It writes results to stdout in CSV format.) Below are the results† of a few runs of the benchmark:

Hardware: 8-core E5520 (Nehalem) @ 2.27GHz (2 sockets at 8 cores each)
Hyperthreading turned off
6GB memory
Windows 7, 64-bit
Compiler: Intel Compiler, version 13.0 beta 2+ at optimization /O3,
Program: 64-bit executable
Total iterations per test: 100,000,000
Each test repeated 5 times
Min-ticks = fastest run for each test. avg.-ticks = average of 5 runs.

name	in-order?	min-ticks	avg.-ticks
Serial	yes	5757	5934.2
Cilk with worker-indexed array	no	1918	2243.2
Cilk with combinable	no	1778	1809.6
Cilk with reducer	yes	1731	1762.8
Cilk with worker-indexed array and inner loop	no	2028	2187.0
Cilk with combinable and inner loop	no	1747	1765.4
Cilk with reducer and inner loop	yes	1763	1781.4

The highlighted regions show that the performance with the reducer is as good as using **Combinable**. Note that the reducer produced results in order and Combinable produced results out of order. The less-than-stellar speed-up (about 3.2x on 8 cores) is probably caused by the extensive memory allocation in the std::list. When I get a chance, I might try this test again using the TBB scalable memory allocator to see if it improves the speedup numbers.

The ability to create code that reads like serial code and produces identical results to serial code is one of the major selling features of Cilk and is a major reason why we would prefer to see a language extension rather than a PPL-like or TBB-like library.

†Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.