Thread Optimization

Overview
This page is about optimizing the software to take full advantage of multithreading across many core CPU's, circa 11/2014

• See Thread Support for basic "thread safety" level issues across the codebase.
• And Vectorizing for SIMD (e.g, AVX) optimizations.

As of Version 8.0, this design is released and in place. It results in major speedups and efficiencies in using multiple CPU threads. See GPU page for a comparison between CPU threads and GPU’s.

Links to Resources
• http://www.ece.unm.edu/~jimp/611/slides/chap5_3.html -- reducing cache misses
• https://gcc.gnu.org/projects/prefetch.html -- gcc prefetch support
• http://www.cc.gatech.edu/~hyesoon/lee_taco12.pdf -- article on software vs. hardware prefetch
• http://linux.die.net/man/3/ numa -- NUMA man page
• http://queue.acm.org/detail.cfm?id=2513149 -- good article on NUMA
• http://stephen-tu.blogspot.com/2013/04/on-importance-of-numa-aware-memory.html -- good benchmark and performance data on NUMA

General Conclusions
• software prefetch is probably not important -- if we organize everything in memory to be contiguous, and thread-local, then the hardware prefetching will work as well or better than software -- software can probably only screw things up
• the implication is that we do NOT need to organize the code in a way that makes it obvious to the compiler what memory is being accessed within a loop -- i.e., we CAN use "opaque" virtual method calls within a loop -- this preserves our ability to use an object-oriented design, through the specs!
• BUT we need to test this out in practice and see how much difference it actually makes..
• GPU computing support however is NOT compatible with virtual function calls, etc. So if we want a more inclusive design, maybe sticking with virtual functions makes more sense?
• thread-local memory is ESSENTIAL. unfortunately NUMA is not avail on mac, but our main compute servers will be linux and it is avail there no prob. not sure about windows.
• there are also "rule of thumb" tricks that we can use outside of NUMA to make the hardware likely to do the right thing anyway.
**Overall Design**

- Each thread must have its own memory for everything it processes
- This memory must be allocated where possible using NUMA allocation routines, and, where not possible, all other constraints in the design must work to encourage local allocation of threads
- Threads must be persistent to the extent possible, because they now own the memory. Hence changes in n_threads require full rebuild.
- Units split into Unit and UnitVars -- latter contains all the compute-relevant variables, former is structural stuff -- only vars are allocated thread-local
- Everything should run through the thread system, and all calls should start at the network level. But we don’t want to write two versions of all the code. Soln is to have every Network method take the thread number as an arg, and use it to access thread-specific memory etc from there.
- `Network::DoSomething(int thread_no)`
Article Sources and Contributors

Thread Optimization  Source: https://gray.colorado.edu/emergent/index.php?oldid=12198  Contributors: Oreilly