Abstract (leave blank for now)

Introduction

- The SSI experience (learning to conduct research & learning about parallelism)

-10 weeks

-on parallelism/research

-“full time” job

-Coding AND research

- TSP problem (references)

-NP complete

-n! possible paths

-

What You Learned

-          TSP Algorithms (sequential & parallel)

-Brute force

-Easy to parallelize on all platforms

-embarrassingly parallel

-finding permutations of paths

-optimizations(?)

-Trimming paths

-          Parallelism in general (hardware, concepts like protecting shared data, splitting up big tasks so multiple processors/cores/nodes can work on it)

-race conditions: They sound scary but are rare and easy to eliminate for certain tasks

-Concurrent memory access: Would be easier to use given a better understanding of how things are stored in memory

-Reducing communications: fairly easy to comprehend, sometimes difficult to weed out.

-Global v. Local v. Shared (for CUDA) memory.

-Cache lines

-          OpenMP

-Easiest to work with, both debugging and coding (Jerome)

-pragmas easy to understand, implement

-works a lot like a for loop

-Requires little software configuration (Jerome)

-“-fopenmp” is not overwhelming

-common hardware setup

-works on multi-core systems as well as the coprocessor! Yay!

-          CUDA

-Hardest to work with, especially debugging (Jerome)

-also requires learning C

-Cannot cout simple things

-Requires moderate configuration to run (linking the appropriate libraries) (Jerome)

-hardest to optimize(Jerome)

-Threads, latency, small caching, etc.

-most difficult to grasp architecture (both memory and thread distribution) (Jerome)

-threads, blocks, grids, warps

-shared v. global GPU memory

-Cannot use many C library (only \_\_global\_\_ and \_\_kernel\_\_)

-nextPermutation()

-          MPI

-Most arduous configuration (Jerome)

-BCCD does most of the software config, but still difficult overall (Zach)

- Coprocessor (openMP)

-Even more difficult to set up than MPI, but more understandable (Jerome)

-this might be because we did MPI first

Experiments with LittleFe & Results

-Better to spread task out over multiple processors rather than an equivalent number of nodes

-Collision of memory access, etc. on the same node

-combining MPI and openMP generally performed more poorly than pure MPI at maximum CPU usage.

-no idea why

-Overall, the hybrid code performed rather inconsistently (pending further trials)

-again, no idea why

-for pure MPI, a better% speedup was found when increasing the problem size

-possibly because the parallelizable portion of the code increases in size

-Overall, expected vs. actual speedup was less favorable as CPU usage increased

-process is perhaps colliding with behind-the-scenes processes

Conclusions

-openMP is a great start to parallel computing

-When we started, we even thought: “this is easy!”

-Parallel computing requires good knowledge of scope and copying of variables

-what accesses create race conditions

-what variables exist as an independent copy for each thread

-Focus on optimization is interesting, but can make parallel computing seem somewhat overwhelming

-limiting thread communication

-limiting thread writes to critical memory

-Consecutive memory accesses

-We had more hardware and config issues than actual coding issues

-program interactions with CUDA

-overheating

-difficulty debugging encourages small steps in working up to complex algorithms

-See: difficulty debugging

-VERY VIABLE TO TEACH PARALLEL COMPUTING EARLY IN THE MAJOR

-No need for objects, scope and arrays are good topics to know, however

Future Work

-intense use of analyzing tools to see what we are doing “wrong”

-Divide thread loads into parcels and distribute them?

-Implement certain optimizations to the algorithm

-optimize the brute-force algorithm for each piece of hardware

Acknowledgements

-Dave Toth (if not already mentioned as an author)

-Ian Finlayson (for providing technical assistance [intentional segfaults!])

-UMW SSI (UMW)

-NVidia

-Intel

References