ECE 574 – Cluster Computing Lecture 11

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

28 February 2017

Announcements

- Homework #6 was posted. Was a problem with the link to the code in the original handout, it has been updated.
- Midterm will be after the break. Still deciding what form it will take.

HW#5 Review

- Low-level C is a pain. Things like passing pointers to double-indexed arrays, and `(void *)` casting.
  I'd like to say you'll never see this, but if you ever get a job doing Linux kernel or similar low level work there's a lot of this that goes on.
- Hopefully you'll find OpenMP is a lot simpler.
- Some results on a 10848x10824 NASA image I found:

bench	Load	convolve	combine	store
before	945,172	20,972,969	1,740,545	865,404
coarse(2)	952,647	10,752,946	1,785,945	882,353
fine 1	960,527	10,582,954	12,303,506	921,339
fine 2		5,418,575	6,255,203	
fine 8	935,998	1,491,921	3,574,811	928,533
fine 16		729,125	2,097,431	
fine 32		627,906	714,431	

OpenMP Examples

See the course website for a link to a tarball with all the examples.

Simple

`openmp_simple.c` just creates a parallel region and prints thread number. By default, how many threasd are set up on the Haswell-EP machine?

Scope

TODO: private/shared variable example

for

`openmp_for.c`

- Parallelizes the memory init loop.
- Thread number set from command line and the `num_threads()` directive.
- What happens to performance as you add threads?

static schedule

`openmp_static_schedule.c`

- Creates 100 threads with chunksize of 1.
- Threads are assigned loop indices at compile time.
- In example, thread 0 is fastest and 4 the slowest.
- You can see thread 0 runs through its assignment fast and then sits around doing nothing while the rest slowly finish.

dynamic schedule

`openmp_dynamic_schedule.c`

- Creates 100 threads with chunksize of 1.
- Threads are assigned loop indices dynamically.
- Each thread starts with one, but zero runs all the rest because it is so fast.

Changing Chunksize

`openmp_dynamic_chunk.c`

- Creates 100 threads with a prime chunksize.
- Threads are assigned same amount of time to run.
- Spread mostly evenly but the last set of chunks, only two threads get assigned while the others have nothing to do.
- Switch to “guided” and the chunksize decreases over time and the ending is a bit more balanced.

critical

`openmp_critical.c`

- Has a parallel loop, but a shared global counter inside.
- What happens without a critical section? (race condition)
- Put in the critical section get right results.
- But slow!
- No need to manually add mutexes, OpenMP abstracts that away.

section

`openmp_section.c`

- For parallelism when you don't have a loop
- Have multiple functions that have no dependencies, want to run at same time?
- No matter how many threads you have, only can run up to the maximum number of sections at a time.

reduction

`openmp_reduction.c`

- What if you calculate something in each loop iteration, but want to sum them all in the end? Something like a vector dot product?
- You could put it in a for loop, $sum = sum + i * a[i]$ but race condition on shared sum.
- Could put in critical section but that's slow as we saw.
- Instead can use special reduction directive.

simd reduction

`openmp_simd_reduction.c`

`https://software.intel.com/en-us/articles/enabling-simd-in-program-using-openmp40`

- simd directive
- Supported by recent GCC (5.0 and later)
- Tries to map your code into SSE/AVX vector instructions if available on your processor.
- Our example turns out runs *slower*. Possibly our input set is not big enough.
- Can look at assembly code to verify it is making SIMD

code:

```
objdump --disassemble-all openmp_simd_reduction
```

- Also you can use `gcc -S` to generate assembly.

offload

Can offload to GPU or MIC.

`https://gcc.gnu.org/wiki/Offloading`

Need separate compiler for component. Support really isn't there yet.