SU3 matrix multiply

Objectives

- Re-order loops for performance improvement
- See effect of `numactl` on performance
- Optimizing number of threads is tricky

SU(3) mutiplication

- Log into qpace3
- source setup file, and load compiler module, if needed
- `cd LATT_PRAC_EXERCISES/03_SU3_MULTIPLY/`
- Three similar source codes all do approximately the same thing: multply an array of complex 3×3 matrices with 3-component complex vectors to get a 3-component complex result:

$$a_x^i = M_x^{ij} b_x^j,$$

where $i, j = 0, 1, 2$, and $x = 0 to V$ where V is some large number we can call "VOLUME'.

SU(3) mutiplication

These are just testbeds to try different strategies in a simple case. Ignore the idiosyncrasies:

- Static allocation, macros
- Potenital math errors, spelling errors, bugs......
- (almost) everything in `main()`
- Your coding and algorithms are more elegant, naturally
- ....

1: math_su3_test.c

Change

```
for(indx=0; indx<VOLUME; indx++){
  for(i=0; i<SU3VEC_SIZE; i++){
     for(j=0; j<SU3VEC_SIZE; j++){
       math happens here
     }
   }

}
```

to

```
for(i=0; i<SU3VEC_SIZE; i++){
   for(j=0; j<SU3VEC_SIZE; j++){
    for(indx=0; indx<VOLUME; indx++){
       math happens here
     }
   }

}
```

- What does the vector report say in each case?
- How does performance change?

2: math_su3_test_blocked.c

Change

```
#pragma omp parallel for private (lo,hi,...)
for(tindex=0; tindex<numthreads; tindex++){
  compute offsets here
  for(indx=lo; indx<hi; indx++){
    for(i=0; i<SU3VEC_SIZE; i++){
       for(j=0; j<SU3VEC_SIZE; j++){
         math happens here
       }
     }
   }

}
```

to

```
#pragma omp parallel for private (lo,hi,...)
for(tindex=0; tindex<numthreads; tindex++){
  compute offsets here
  for(i=0; i<SU3VEC_SIZE; i++){
     for(j=0; j<SU3VEC_SIZE; j++){
      for(indx=lo; indx<hi; indx++){
         math happens here
       }
     }
   }

}
```

- What does the vector report say in each case?
- How does performance change?
- What bandwidth is achieved in the kernel?

2: math_su3_test_blocked.c — Memory bandwidth

Bind the executable to the High Bandwidth memory. Look at numa.txt from first exercise.
Try changing the batch file so the executable line is

```
numactl -m 4-7 /complete-path-to-executable/math_su3_test_blocked
```

Also try to maximize performance through adjusting OMP_NUM_THREADS

- What maximum bandwidth do you see?
- Is the maximum performance at an integer number of threads/core?

3:`math_su3_test_unroll.c` — Arithmetic Intensity

When computation is “bandwidth limited”, the memory cannot feed the processor fast enough to sustain the processor’s peak flops/sec rate.
Can we increase the

$$\text{Arithmetic intensity} = I = \frac{\text{flops}}{\text{byte}}$$

to improve performance?
Try: store only 1st two columns of $SU(3)$ matrix. Reconstruct the 3rd column on the fly.

- more flops will be needed per iteration
- less data will be needed per iteration

Will time to solution decrease?

3: `math_su3_test_unroll.c` — Arithmetic Intensity

- Look at the matrix multiplication in
  `math_su3_test_unroll.c`
- i, j loops are unrolled within the *VOLUME* loop
- Defining `FORMAT_2COL` for the preprocessor uses 2-col format
- Make both
  `math_su3_test_unroll`
  and
  `math_su3_test_unroll_2col`
  (differ by definition in makefile).
- Submit and compare elapsed time to solution.