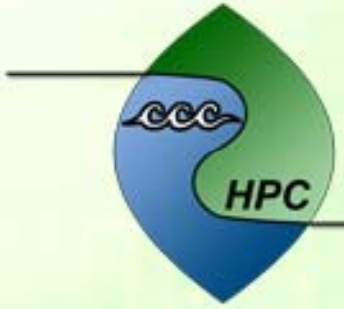
OpenMP

Shared Memory Interface

Tom Murphy

Director of Contra Costa College
High Performance Computing Center



Preliminaries

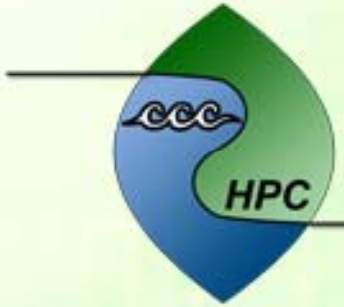
What is OpenMP?

- Enables shared memory parallelism
- Consists of
  - Compiler directives
  - Functions
  - Environment variables
- Requires a supportive compiler
- C, C++, and Fortran
  - Are the languages of OpenMP
  - We will be using C

Preliminaries

which version are we using?

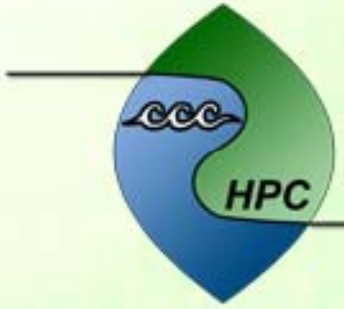
- OpenMP 2.5
  - Gcc 4.2 (May 2007) supports OpenMP 2.5
  - When Gcc 4.4 releases it will support 3.0



Preliminaries

How do we use it?

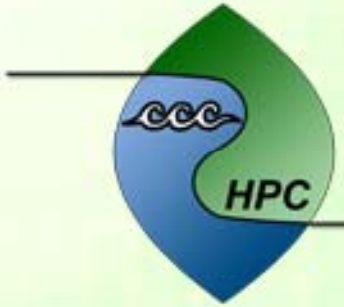
- To setup to run the compiler
  - alias ompcc='icc -openmp -openmp-report2’
- You can now use ‘ompcc’
  - In place of ‘icc’ or ‘gcc’



Sequential Hello world

the code of “hello.c”

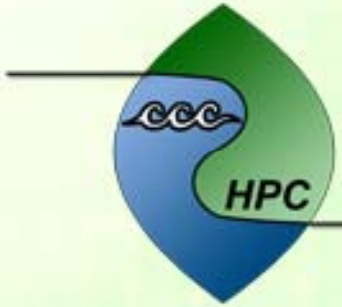
```
#include <stdio.h>
int main () {
   //#pragma omp parallel
   //{
   printf("Hello World!\n");
   //}
   return 0;
}
```



Sequential Hello world

starting at the beginning is interesting

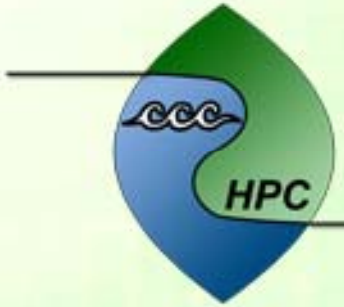
- icc hello.c
  - ./a.out
  - No surprises
- ompcc hello.c
  - What do you expect?
- Uncomment comments: expecting?
  - icc hello.c
  - ompcc hello.c



Simplest OpenMP example?

the code of “for.c”

```
#include <omp.h>
#include <stdio.h>
int main () {
    int i;
   #pragma omp parallel for
    for(i=0; i<10; ++i) {
            printf("i=%d\n", i);
    }
    return 0;
}
```

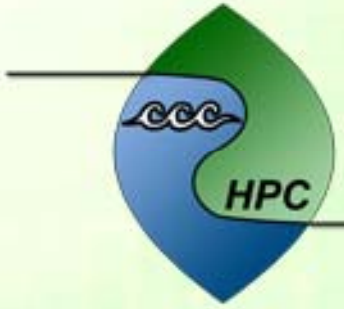


Simplest OpenMP example?

parallelizing a for loop

- Run the command “ompcc for.c”
- Run the command “icc for.c”
- _OPENMP should be defined
- Split printf into two lines

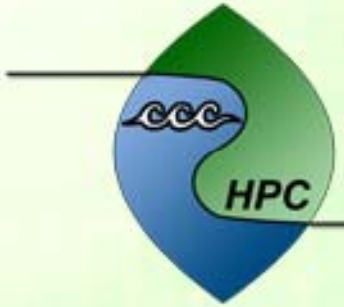
```
printf("i=“);
printf(“%d\n", i);
```



Sharing is not always good

the code of “ranksize.c”

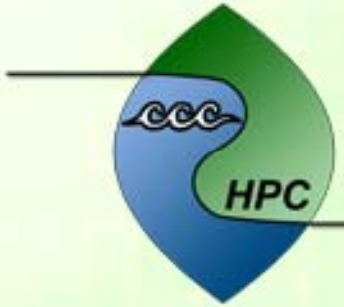
```
#include <omp.h>
#include <stdio.h>
#define WORKLOAD 1
int main () {
      int rank, size, i;
      #pragma omp parallel
      {
            rank = omp_get_thread_num();
            for(i=1; i<WORKLOAD; ++i);
            printf("Hello World from thread %d\n", rank);
            if ( rank == 0 ) {
                  size = omp_get_num_threads();
                  printf("There are %d threads\n",size);
            }
      }
      return 0;
}
```



Sharing is not always good

lots of key things happen now

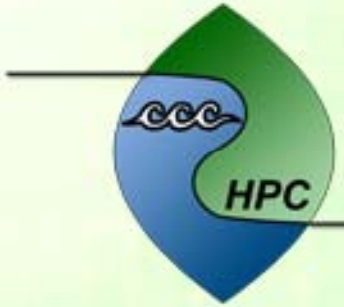
- Run “icc ranksize.c”
  - Can _OpenMP still help?
- ompcc ranksize.c
  - Run it several times
  - Change WORKLOAD to be 1000000
- We need a separate copy of rank in each thread
  - Add “private(rank)” clause to pragma “parallel “
  - Why didn’t the variable “I” in “for.c” fail us?
  - Are we done?



How to measure success?

Lower wallclock or efficient CPU use?

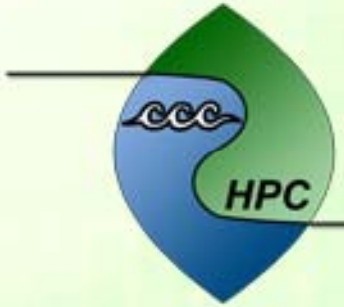
- Wall clock is easy to measure
  - It's what the user cares about
- CPU use is harder to measure
  - It's what the data center cares about
  - Profiling tools exist, and are important
- Close enough is also success
  - Human time is also valuable



It's all about timing

the code of "timing.c"

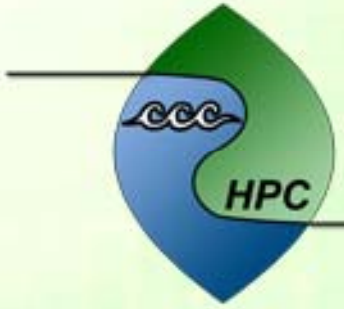
```
#include <omp.h>
#include <stdio.h>
#define WORKLOAD 1
#define MAXDIM 10
int main () {
      int i, wl;
      double a[MAXDIM], b[MAXDIM], c[MAXDIM];
      for(i=0;i<MAXDIM;++i) a[i]=b[i]=c[i]=i;
    #pragma omp parallel for  private(wl)
      for(i=0;i<MAXDIM;++i) {
              for(wl=0;wl<WORKLOAD;++wl) c[i] *= a[i]/b[i];
      }
      for(i=0;i<MAXDIM;++i) printf("%d:\t%f\n", i, c[i]);
      return 0;
}
```



It's all about timing

can see effect of parallelization overhead

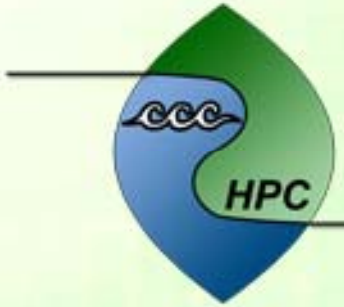
- "time ./a.out" gives overall wallclock time
- double omp_get_wtime(void)
  - Gives more fine grained control
- Requires some code changes to use it
  - Split "parallel for" into two pragmas
  - Create variable "deltaT" in scalar part
  - Calculate deltaT at top and bottom
  - Do a reduction on "deltaT"



It's all about timing

little more detail on the changes

- Split "parallel for" into two pragmas
  - #pragma omp parallel
  - #pragma omp for private(wl)
- Create variable "deltaT" in scalar part
  - double deltaT;
- Calculate deltaT at top and bottom
  - deltaT = omp_get_wtime();
  - deltaT = omp_get_wtime() - deltaT;
- Do a reduction on "deltaT" (first pragma)
  - #pragma omp parallel reduction(+:deltaT)

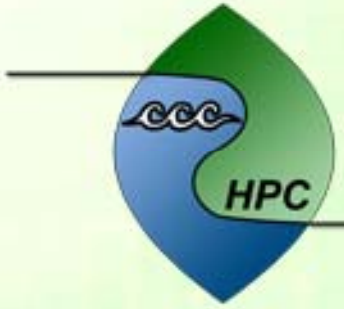


All about reductions

lots of possibilities

- Specify operator and a list of variables
  - Can have more than one clause, as needed
  - Private copy made, initialized relative to operator
- Operator initial value

+	0
-	0
*	1
&	~0
|	0
&&	1
||	0

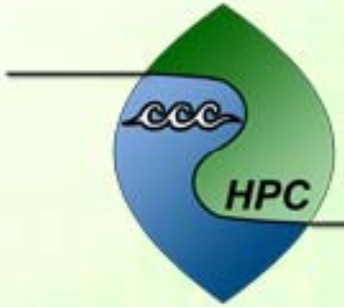


Simplest revisited

exploring for.c and timing changes

- Printf in two pieces didn't print together
- This is critical - add the right pragma

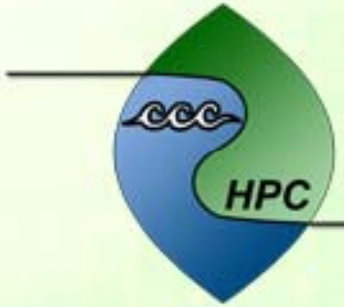
```
pragma omp critical (printTest)
{
   printf("i=“);
   printf("%d\n", i);
}
```



Simplest revisited

but they are still out of order

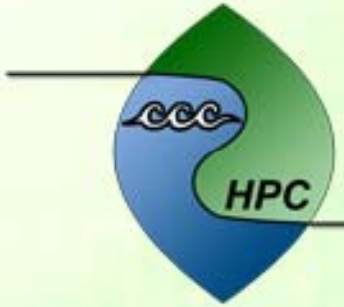
- Let's force iterations to be in sequence
- Add "ordered" as clause on "parallel for"
- Use timing calls to understand
  - before and after costs of being ordered



timing loop revisited

we can control scheduling

- Four possible for clauses
  - schedule(static, iterations/numthreads)
  - schedule(dynamic, 1)
  - schedule(guided, 1)
  - schedule(runtime)
    - OMP_SCHEDULE envar
    - OpenMP 3.0 gives better runtime control
- Modify timing.c and time differences
  - Make work loop go to i*WORKLOAD
  - Make work loop go to (MAXDIM-I)*WORKLOAD



Additional experiments

to run in your copious spare time

- OMP_NUM_THREADS
  - Allows you to set the number of threads to use
    - void omp_set_num_threads(integer)
    - Integer omp_get_num_threads()
- Create a temporary array
  - make it bigger and/or more threads
  - When do things destabilize?
  - How can you know?
  - OMP_STACKSIZE comes with OpenMP 3.0