Programming Project

I. Project Description
II. Programming Concepts
III. Parallelization Strategies
IV. Implementation Details
I. Project Description

• We want to compute π
• One method: method of darts*
• Ratio of area of square to area of inscribed circle proportional to π

*Disclaimer: this is a TERRIBLE way to compute π. Don’t even think about doing it this way except for the purposes of this project!
Method of Darts

- Imagine dartboard with circle of radius $R$ inscribed in square
- Area of circle = $\pi R^2$
- Area of square = $(2R)^2 = 4R^2$
- Area of circle
  Area of square = $\frac{\pi R^2}{4R^2} = \frac{\pi}{4}$

Method of Darts

- So, ratio of areas proportional to $\pi$
- How to find areas?
  - Suppose we threw darts (completely randomly) at dartboard
  - Could count number of darts landing in circle and total number of darts landing in square
  - Ratio of these numbers gives approximation to ratio of areas
  - Quality of approximation increases with number of darts
- $\pi = 4 \times \frac{\# \text{ darts inside circle}}{\# \text{ darts thrown}}$
Method of Darts

• Okay, Rebecca, but how in the world do we simulate this experiment on computer?
  – Decide on length $R$
  – Generate pairs of random numbers $(x, y)$ s.t.
    $-R \leq x, y \leq R$
  – If $(x, y)$ within circle (i.e. if $x^2 + y^2 \leq R^2$), add one to tally for inside circle
  – Lastly, find ratio

II. PROGRAMMING CONCEPTS

II. Programming Concepts

• Pseudorandom numbers
• Typecast and coercion
• Datatypes

Pseudorandom Numbers

• In C language, function int `rand(void)` generates “pseudo-random integer in range 0 to RAND_MAX”
• `RAND_MAX`: C-language constant denoting maximum random number generated; actual value varies with implementation
• Divide “random” number by maximum random number to get a number between 0 and 1*
• Numbers generated by `rand()` not really random; same sequence every time
• Change seed for random number generator with `void srand(unsigned int seed)`

*Disclaimer: this is a TERRIBLE way to compute a pseudorandom number. Don’t even think about doing it this way except for the purposes of this project!
Type Cast and Coercion

• \texttt{int a = rand(); double b = a/RAND\_MAX;}  
  \quad - b equals 0
• \texttt{int a = rand(); double b = ((double) a)/(double)\ RAND\_MAX);}  
  \quad - b equals correct value

Type conversion rules:
  - \texttt{int/int \rightarrow int}
  - \texttt{int/double \rightarrow double}
  - \texttt{double/int \rightarrow double}
  - \texttt{double/double \rightarrow double}

Datatypes

• For large number of darts, need larger datatype than \texttt{int} or risk overflow

• On some computers (varies by platform):

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{int}</td>
<td>-32,768 \rightarrow +32,767</td>
</tr>
<tr>
<td>\texttt{long int}</td>
<td>-2,147,483,648 \rightarrow +2,147,483,647</td>
</tr>
<tr>
<td>\texttt{unsigned long int}</td>
<td>0 \rightarrow +4,294,967,295</td>
</tr>
</tbody>
</table>
III. Parallelization Strategies

• What tasks independent of each other?
• What tasks must be performed sequentially?
• Using PCAM parallel algorithm design strategy
Partition

“Decompose problem into fine-grained tasks to maximize potential parallelism”

Finest grained task: throw of one dart

Each throw independent of all others

If we had huge computer, could assign one throw to each processor

Communication

“Determine communication pattern among tasks”

- Each processor throws dart(s) then sends results back to manager process
Agglomeration

“Combine into coarser-grained tasks, if necessary, to reduce communication requirements or other costs”

• To get good value of $\pi$, must use millions of darts
• We don’t have millions of processors available
• Furthermore, communication between manager and millions of worker processors would be very expensive
• Solution: divide up number of dart throws evenly between processors, so each processor does a share of work

Mapping

“Assign tasks to processors, subject to tradeoff between communication cost and concurrency”

• Assign role of “manager” to processor 0
• Processor 0 will receive tallies from all the other processors, and will compute final value of $\pi$
• Every processor, including manager, will perform equal share of dart throws
IV. IMPLEMENTATION DETAILS

Detail from Vincent van Gogh's Sunflowers. Source:
http://painting.about.com/od/famouspainters/ig/Van-Gogh-and-Expressionism/Sunflower-Detail.htm

IV. Implementation Details

1. Implement using six basic MPI functions
2. Add OpenMP capabilities
3. Implement using collective operations
Step 1

- Create function \texttt{pi\_basic(...)} that uses only six basic functions covered in part 1
  - \texttt{pi\_basic(...)} should call function \texttt{throw\_darts(...)} to perform the actual throwing of darts
- Test your implementation and make sure it works

Step 2

- Use OpenMP to parallelize \texttt{throw\_darts(...)} over a node
- Parallelization will occur in loop
- Make sure code works properly
Step 3

- Create function `pi_advanced(...)` that uses MPI collective operations
- This should require trivial change from `pi_basic(...)`

Skeleton Code

```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char **argv) {
    /* declarations here */
    MPI_Init(&argc, &argv);
    double start = MPI_Wtime();
    pi_simple(...);
    double finish = MPI_Wtime();
    printf("Processor \%d took \%f s for pi_simple", me, finish-start);
    double start = MPI_Wtime();
    pi_advanced(...);
    double finish = MPI_Wtime();
    printf("Processor \%d took \%f s for pi_advanced", me, finish-start);
    MPI_Finalize();
    return 0;
}
```
Doing this Project on Smoky

• Bring up shell on Mac or Linux or PuTTY shell on Windows
• Log into jaguar with your username (temporary guest accounts or your regular account)
  – ssh -Y hqi@smoky.ccs.ornl.gov
  – Enter your PIN number and then 6-digit SECURID number
• Create directory for program and write program
• Compile using mpicc (e.g. mpicc -o pi.o pi.c)
  – For OpenMP, use -mp=nonuma flag, i.e., mpicc -mp=nonuma pi.c
• No link to MPI or OpenMP libraries necessary – Smoky takes care of that
• Write batch script and submit using qsub scriptname

Bibliography/Resources

• C: The float and double Data Types and the sizeof Operator http://www.iota-six.co.uk/c/b3_float_double_and_sizeof.asp
• C Data types http://www.phim.unibe.ch/comp_doc/c_manual/C/CONCEPT/data_types.html
• OLCF Webpages http://olcf.ornl.gov/
Appendix: Better Ways to Compute \( \pi \)

- Look it up on the internet, e.g. [http://oldweb.cecm.sfu.ca/projects/ISC/data/pi.html](http://oldweb.cecm.sfu.ca/projects/ISC/data/pi.html)
- Compute using the BBP (Bailey-Borwein-Plouffe) formula
  \[
  \pi = \sum_{n=0}^{\infty} \left( \frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \left( \frac{1}{16} \right)^n
  \]
- For less accurate computations, try your programming language's constant, or quadrature or power series expansions

Appendix: Better Ways to Generate Pseudorandom Numbers

- For serial codes
  - Mersenne twister
  - GSL (Gnu Scientific Library), many generators available (including Mersenne twister) [http://www.gnu.org/software/gsl/](http://www.gnu.org/software/gsl/)
- For parallel codes
  - SPRNG, regarded as leading parallel pseudorandom number generator [http://sprng.cs.fsu.edu/](http://sprng.cs.fsu.edu/)