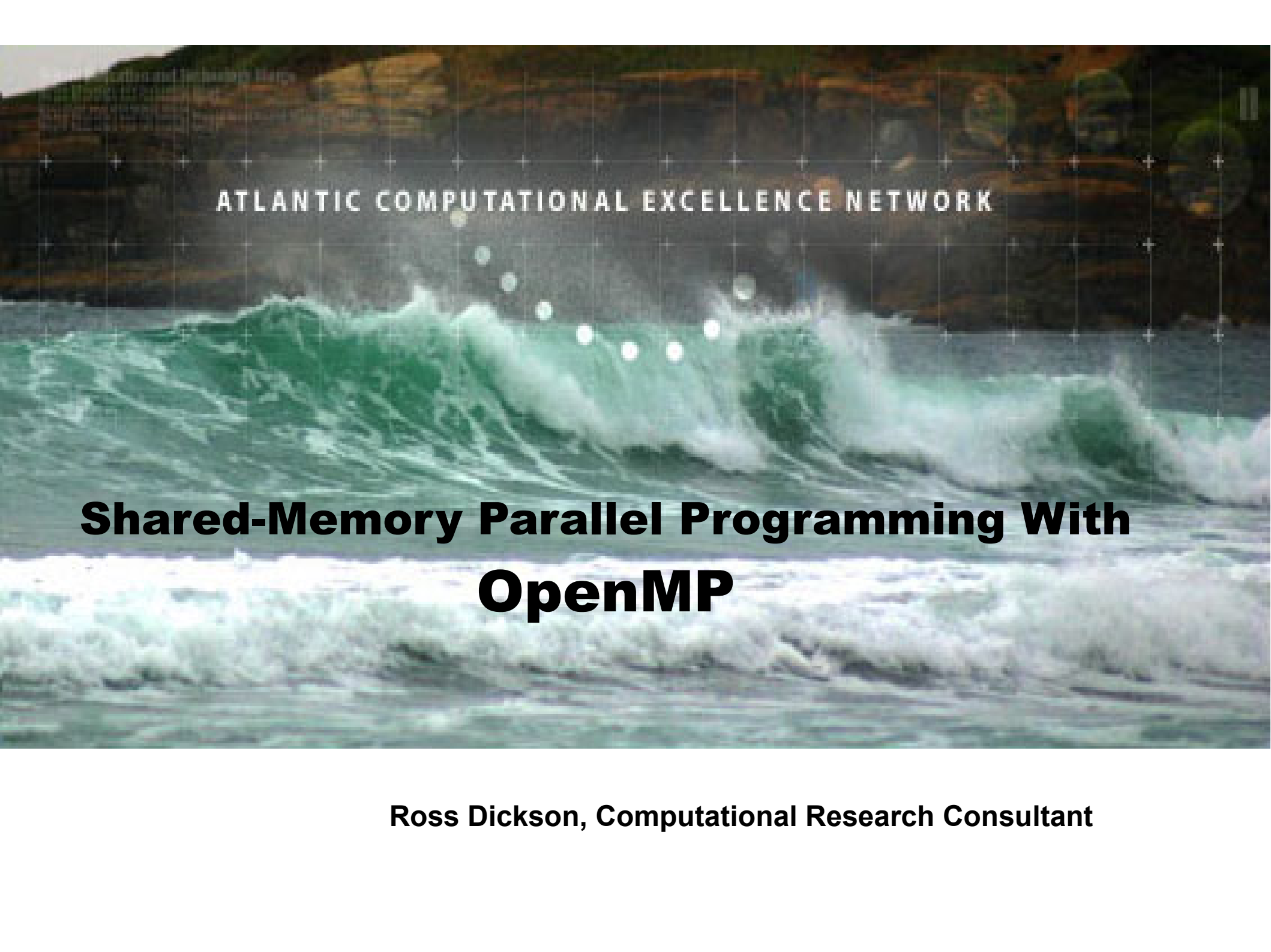




ATLANTIC COMPUTATIONAL EXCELLENCE NETWORK

Shared-Memory Parallel Programming With OpenMP

Ross Dickson, Computational Research Consultant

What is OpenMP?

- Open Multiprocessing **API**
 - First published 1997, v.3.0 published 2008
- A **standard** for shared-memory programming
 - with support from: AMD, Cray, Fujitsu, HP, IBM, Intel, Portland Group, SGI, Oracle, MS, TI; also ANL, LLNL, EPCC, NASA...
 - Fortran, C and C++ specifications in standard
- Shared-memory $\approx$ inside one multi-core machine
- Allows incremental parallelization

Want to follow along?



```
$ cp /home/rdickson/demo/OpenMP/* .  
$ ls  
hello      mandel      reduce.f95  saxpy.f90  
hello.c    mandel.f90  reduce.in   saxpy.in  
job.sh     reduce     saxpy
```

A Simple Loop

saxpy.f90

```
subroutine saxpy(a, x, y, n)
integer i, n
real a, x(n), y(n)
!$omp parallel do
do i = 1, n
    y(i) = a * x(i) + y(i)
end do
return
end
```

- Most work done with compiler directives:
 - `!$omp directive` in Fortran
 - `#pragma omp directive` in C, C++

Compilation

- OpenMP supported by these compilers at ACEnet:
 - Portland Group (PGI)
 - Intel
 - GNU version 4 and later
 - SunStudio
- Different flags for each compiler suite:

```
$ pgf90      -mp      saxpy.f90
$ ifort      -openmp   saxpy.f90
$ gfortran   -fopenmp  saxpy.f90
$ f90        -openmp   saxpy.f90
```

Execution

- Environment variable `OMP_NUM_THREADS` controls number of threads

```
$ export OMP_NUM_THREADS=3
$ ./hello
Hello World from thread 0
Hello World from thread 2
Hello World from thread 1
There are 3 threads
$
```

bash syntax shown; in tcsh use “setenv `OMP_NUM_THREADS` 3”

Use with Grid Engine

- Best practice: Set OMP_NUM_THREADS from SGE variable NSLOTS

job.sh

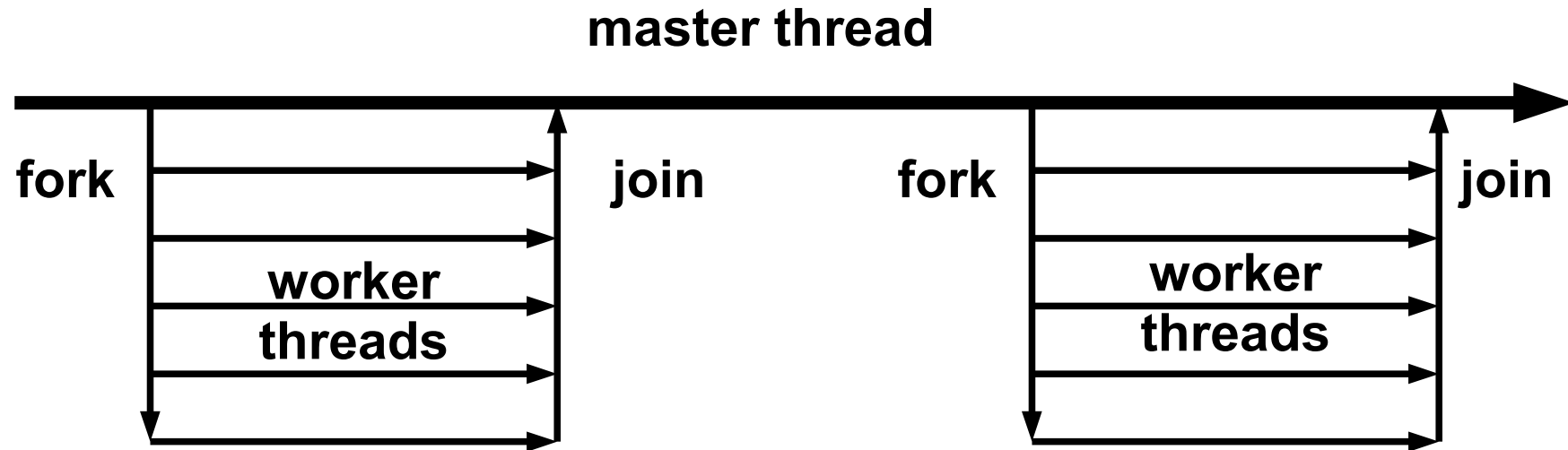
```
## -cwd
## -j y
## -l h_rt=0:1:0,test=true
## -pe openmp 4
export OMP_NUM_THREADS=$NSLOTS
./hello
```

Key Programming Concepts



- Threads, fork, join
- Shared vs. private variables
- Reduction
- Data dependence
- Parallel regions
- Scheduling

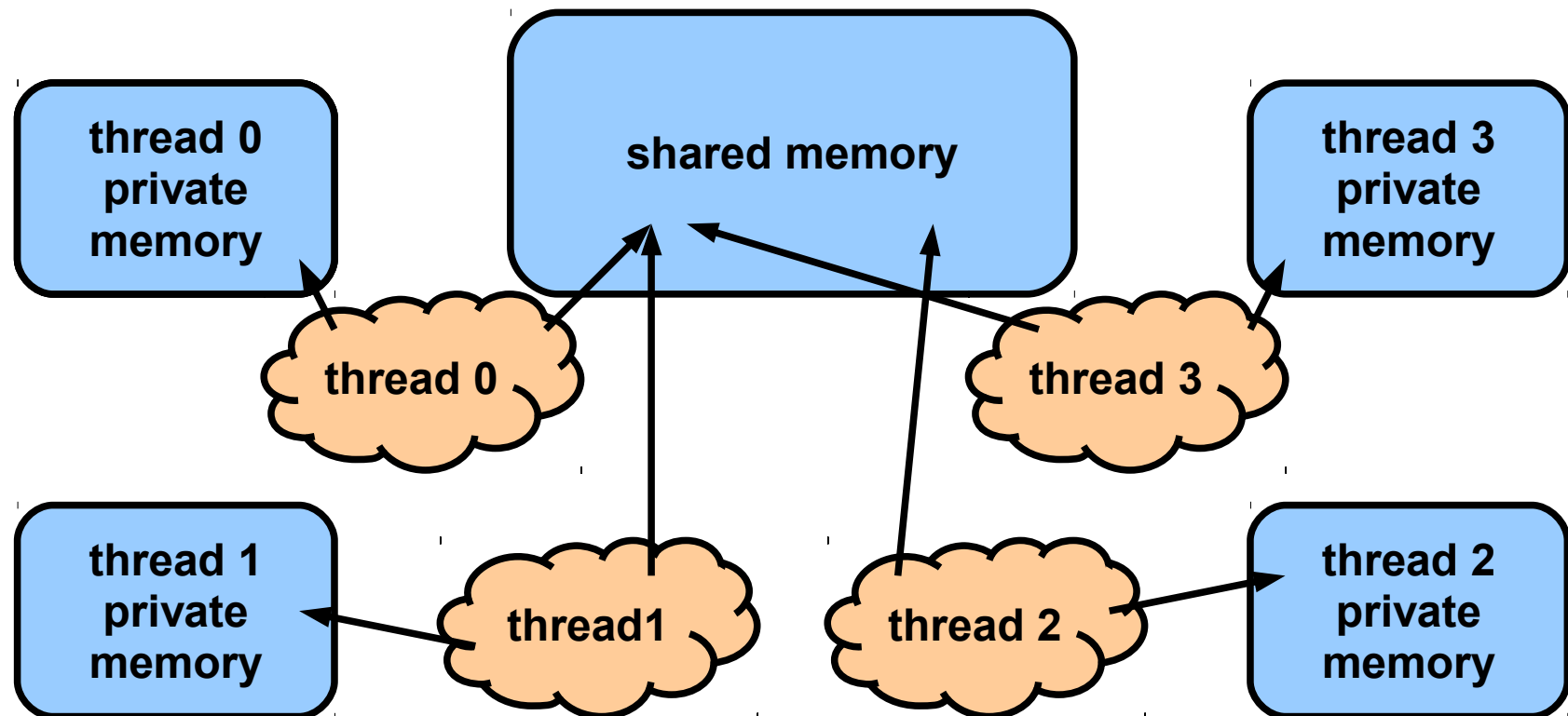
Threads



- Master thread “forks” (creates) worker threads when code enters a parallel section
- Threads “join” (vanish) on leaving a parallel section (*or at least we must assume so*)

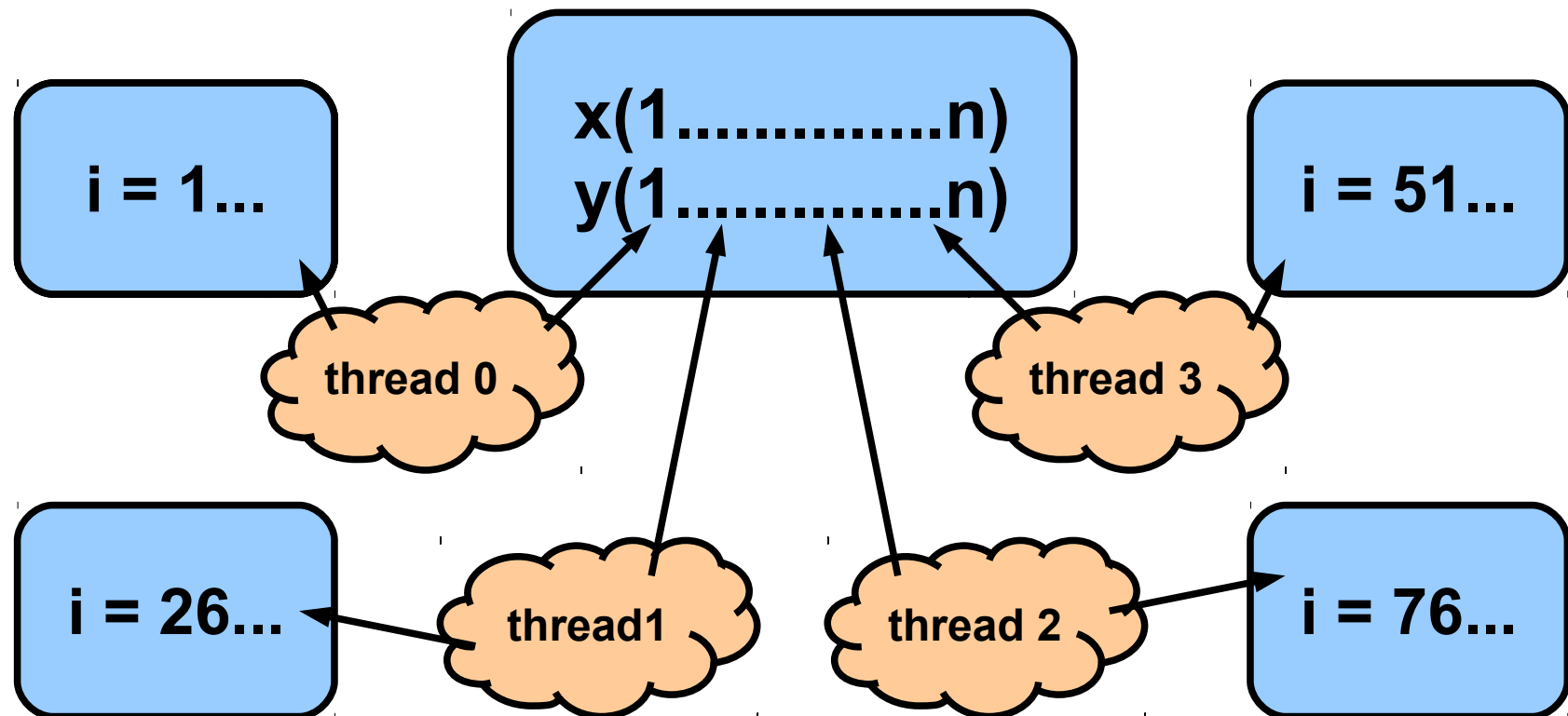
Shared vs. private memory

- Whole program has a shared memory area
- Each thread created with private memory



Shared vs. private memory

- Consider “saxpy” example:
Array index “i” must be private to each thread



Shared vs. private variables

- There are default rules for **scope**
- Most variables shared by default
 - except Fortran loop indices
 - C “for” construct is more complicated
- Most private vars must have scope declared
 - Function (“stack”) locals are private
 - You *can* scope everything explicitly – maybe a good idea?
- See references for details...

Mandelbrot set example

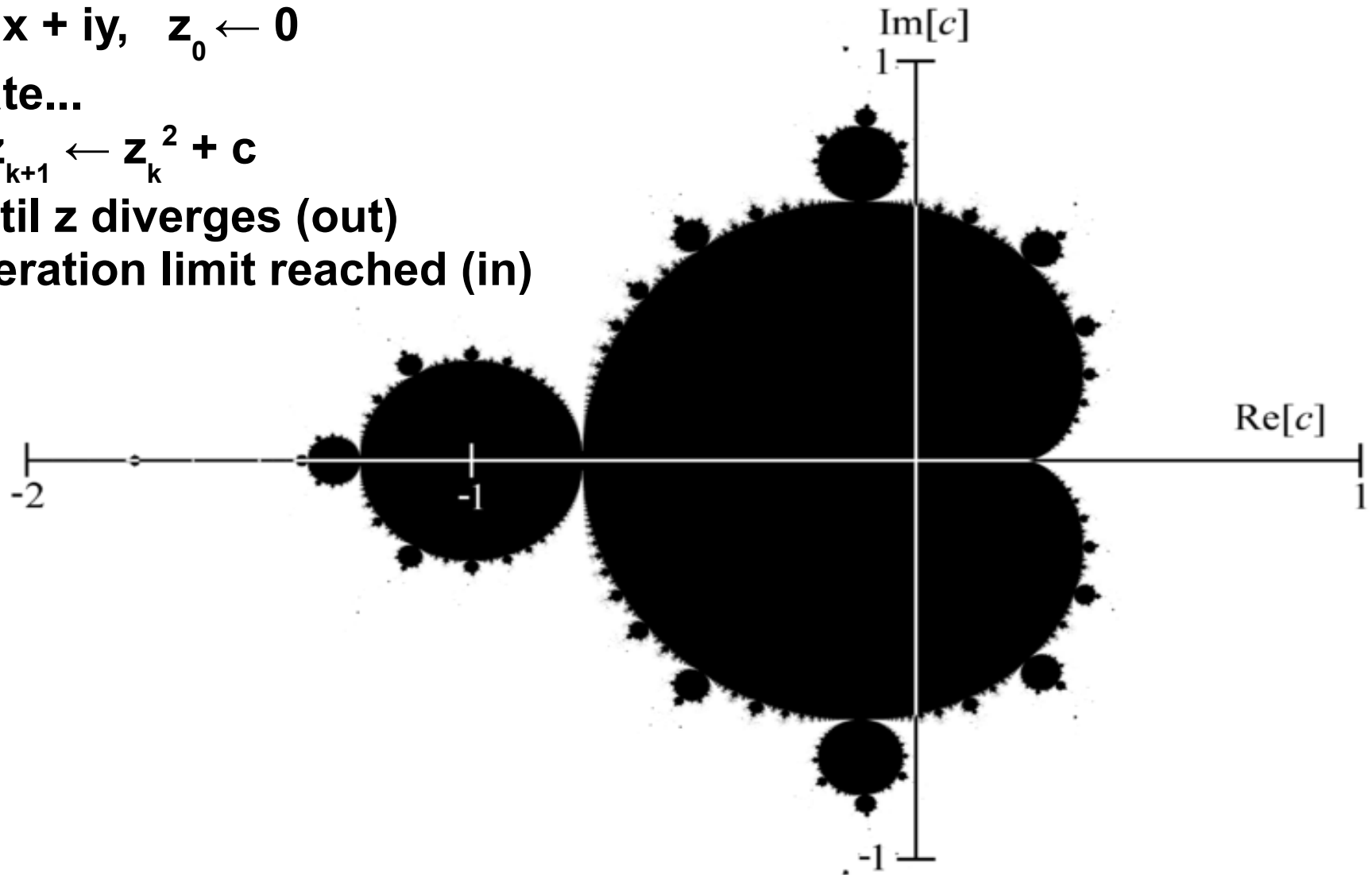
For each point (x, y) :

$$c \leftarrow x + iy, \quad z_0 \leftarrow 0$$

Iterate...

$$z_{k+1} \leftarrow z_k^2 + c$$

...until z diverges (out)
or iteration limit reached (in)



Nested loops

mandel.f90

```
!$omp parallel do private(j, x, y)
do i = 1, m
  do j = 1, n
    x = xmin + i*(xmax-xmin)/m
    y = ymin + j*(ymax-ymin)/n
    depth(j, i) = mandel_val(x, y, maxiter)
  end do
end do
```

- Usually best to parallelize outermost loop → 'i' private
- What about 'j' ?
- 'j' must be private too
- as must 'x' and 'y'
- 'mandel_val' should be a *pure function*

Summing

reduce.f90

```
accumulator = 0.0
!$omp parallel do reduction(+:accumulator)
do i = 1, n
    accumulator = accumulator + x(i)
end do
print *, accumulator
```

- Should `accumulator` be shared or private?
- Answer: Neither. It is a *reduction variable*.
- Reduction operators: sum, product, max, min, logical *or*, bitwise *or*, logical *and*, ...
- OpenMP controls access to reduction variable so threads don't suffer **race condition**

Data Dependence

If one statement writes a memory location,
and another statement either reads or writes
the same location,
then there is a **data dependence** between the
two statements.

- The order in which they are executed affects program correctness...
- ...and therefore affects parallelizability.

Data Dependence examples

```
do i = 2, n
    a(i) = a(i) + a(i-1)
end do
```

- Cannot parallelize this loop:
a(i-1) must be written in iteration (i-1)
before it is read in iteration (i).

```
do i = 2, n, 2
    a(i) = a(i) + a(i-1)
end do
```

- Can parallelize this loop:
Even-numbered elements written;
odd-numbered elements read – no dependence

Data Dependence examples

- What about this one?

```
do i = 1, n
    a(idx(i)) = a(idx(i)) + b(idx(i))
end do
```

- Depends on array **idx**
 - If it's a *permutation*, then no dependence
 - If it contains duplicate entries, then not okay
 - Compiler can't tell!
- Also: Beware common, module, global or static vars inside functions & subroutines!

Parallel Regions

```
integer myid, nthreads
nthreads = omp_get_num_threads()

!$omp parallel private(myid)

myid = omp_get_thread_num()
call do_different_things( myid, nthreads )

!$omp end parallel
```

- Can parallelize general code, not just loops
- Probe OpenMP with functions

Load Balance and Scheduling

```
!$omp parallel do private(j, x, y) &  
!$omp                schedule(dynamic,10)  
do i = 1, m  
    do j = 1, n  
        x = xmin + i*(xmax-xmin)/m  
        y = ymin + j*(ymax-ymin)/n  
        depth(j, i) = mandel_val(x, y, maxiter)  
    end do  
end do
```

- Which loop iterations are assigned to which thread may affect performance – Idle threads inefficient!
- **schedule** clause tells OpenMP how to balance load

Not to mention...

- Types of data dependence
 - and ways to fix some of them
- firstprivate, lastprivate, copyin
- do if, nowait
- Synchronization
 - !\$omp critical, barrier, atomic, ordered
- Work-sharing
 - !\$omp sections, single, master

http://openmp.org

OpenMP.org - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://openmp.org/wp/

Most Visited W Wikipedia B Bookmarks A ACEnet Wiki

ACEnet Support Ticketing System :: Da... OpenMP.org W OpenMP - Wikipedia, the free encyclop...

OpenMP™

THE OPENMP API SPECIFICATION FOR PARALLEL PROGRAMMING

 Subscribe to the News Feed

- »» [OpenMP Specifications](#)
- »» [About OpenMP](#)
- »» [Compilers](#)
- »» [Resources](#)
- »» [Discussion Forum](#)

Events

Input Register
Alert the OpenMP.org webmaster about new products, events, or updates and we'll post it here.
»» webmaster@openmp.org

Search OpenMP.org
Google™ Custom Search

OpenMP News

»SPEC Looking For A Few Good Applications



SPEC, the **Standard Performance Evaluation Corporation**, is looking for realistic OpenMP applications to include in the next version of the SPEC CPU and SPEC OMP benchmark suites.

SPEC is sponsoring a search program, and for each step of the process that a submission passes, SPEC will compensate the Program Submitter (in recognition of the Submitter's effort and skill). A submission that passes all of the steps and is included in the next SPEC CPU benchmark suite will receive \$5000 US overall and a license for the new benchmark suite when released. Details on the Benchmark Search Program at: <http://www.spec.org/cpuw6/>.

Posted on May 20, 2010

»IWOMP 2010: International Workshop on OpenMP



6th International Workshop on OpenMP, June 14-16, 2010, Tsukuba, Japan

"Beyond Loop Level Parallelism in OpenMP: Accelerators, Tasking and More"

The **International Workshop on OpenMP** is an annual series of workshops dedicated to the promotion and advancement of all aspects focusing on parallel programming with OpenMP. OpenMP is now a major programming model for shared-memory systems from multi-core...

The OpenMP API

supports multi-platform shared-memory parallel programming in C/C++ and Fortran. OpenMP is a portable, scalable model with a simple and flexible interface for developing parallel applications on platforms from the desktop to the supercomputer.

»» [Read about OpenMP.org](#)

Get

»» [OpenMP specs](#)

Use

»» [OpenMP Compilers](#)

Learn



References

- <http://www.openmp.org>
- <http://www.ace-net.ca/wiki/OpenMP>
for local documentation
- Examples drawn from Rohit Chandra *et al.*, *Parallel Programming in OpenMP* (Academic Press, 2001)
- Michael J Quinn, *Parallel Programming in C with MPI and OpenMP* (McGraw-Hill, 2004)
- Example code in
`/home/rdickson/demo/OpenMP/*`