OPENMP Performance Issues

OpenMP Overhead

OpenMP overhead due to
(i) Threads spawned/woken up
(ii) Size of Chunks determined
(iii) Threads assigned for dynamic/guided scheduling
(iv) Barriers and reduce operations implemented

OpenMP Speedup

$$S_{openmp} = \frac{1}{(s + (1-s)/\mathrm{N} + \kappa\,\mathrm{N} + \lambda)},$$

$s = serial\ fraction$

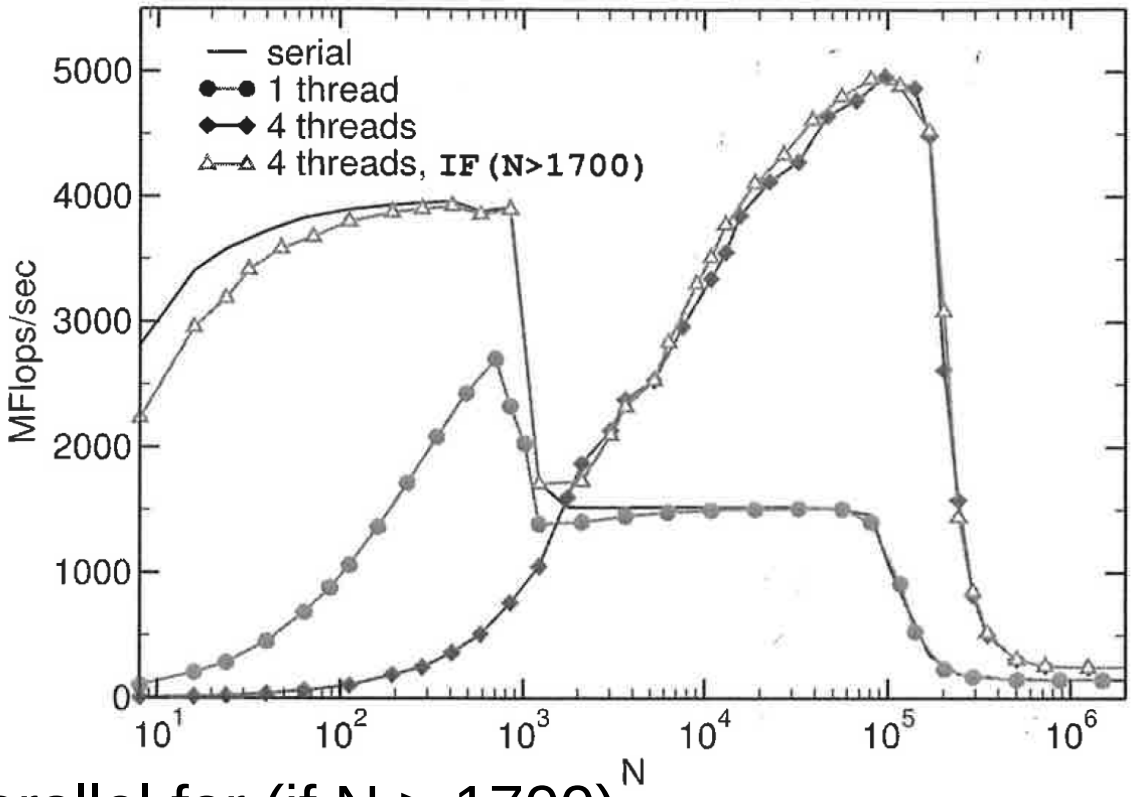
$\kappa = number\ of\ threads\ startup$

$\lambda = other\ startup\ overhead$

Not promising
but depends on
values of constants

OpenMP Overhead

If N is small the overhead of OpenMP will dominate and so the If clause may be used to disable OpenMP for small N

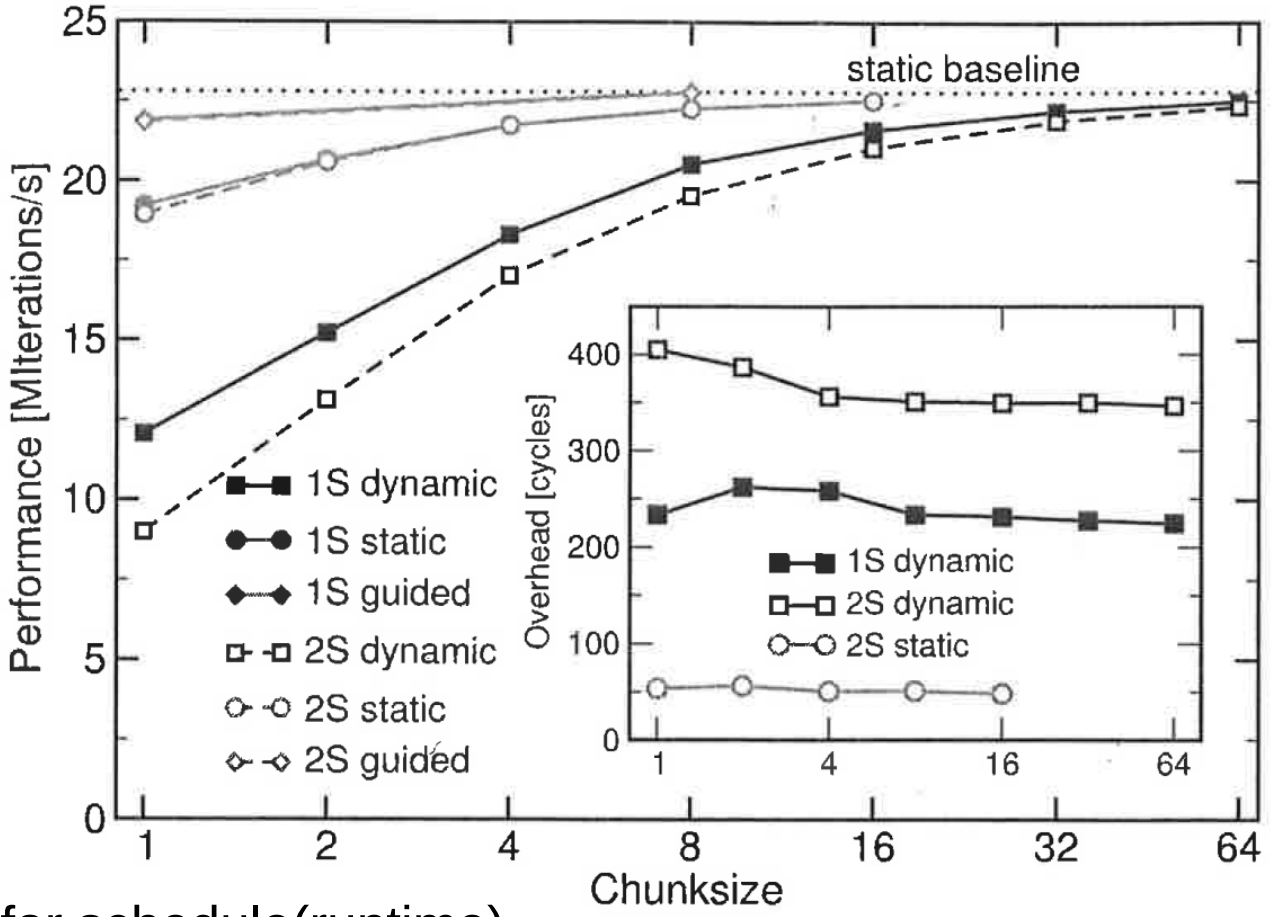



```
# pragma omp parallel for (if N > 1700)
   for(i=1, i<N ,i++)
     {
       a(i) = b(i)+c(i)+d(i);
     }
```

Alternative is to reduce number of threads

Avoid Dynamic Loop scheduling or Tasking

Insert is startup cost
In processor cycles
For assigning a single chunk to a thread




```
# pragma omp parallel for schedule(runtime)
reduction(+:s)
   for(i=1, i<N ,i++)
    {
      s = s + compute(i);
    }
```

Dynamic vs static vs guided on 2 socket 2 core Intel Xeon s=one socket, 2s = two sockets

OpenMP Loop overhead

```
 # pragma omp parallel private (j)
for (j = 1, j < niter, j++)
{ # pragma omp parallel schedule(static) nwait
   for(i=1, i<N ,i++)
     {   a(i) = b(i)+c(i)+d(i);   }
 }
```

Performance Model

$$T(N,\mathrm{nt}) = T_{comp}(N,nt) + T_{setup}(N,nt)$$

$$T_{comp}(N,nt) = \frac{2N}{ntP_s(N/nt)}, \quad \text{where } P_s(N) \text{ is serial perf.}$$

$$T_{setup}(N,nt) = \text{latency} + \text{thread\_part} = T_1 + T_p(t)$$

$$P(N,nt) = \frac{2N}{T(N/nt)}, \quad \text{where } P(N,\text{nt}) \text{ is parallel perf.}$$

$$P(N,nt) = \frac{2N}{2N(ntP_s(N/nt))^{(-1)} + T_{setup}(nt)}$$

Use approximation

$$P_s(N) = \frac{2N}{2N/P_{\max} + T_0}, \; P_{\max} \text{ is asymptotic serial perf. } T_0 \text{ is serial o/h}$$

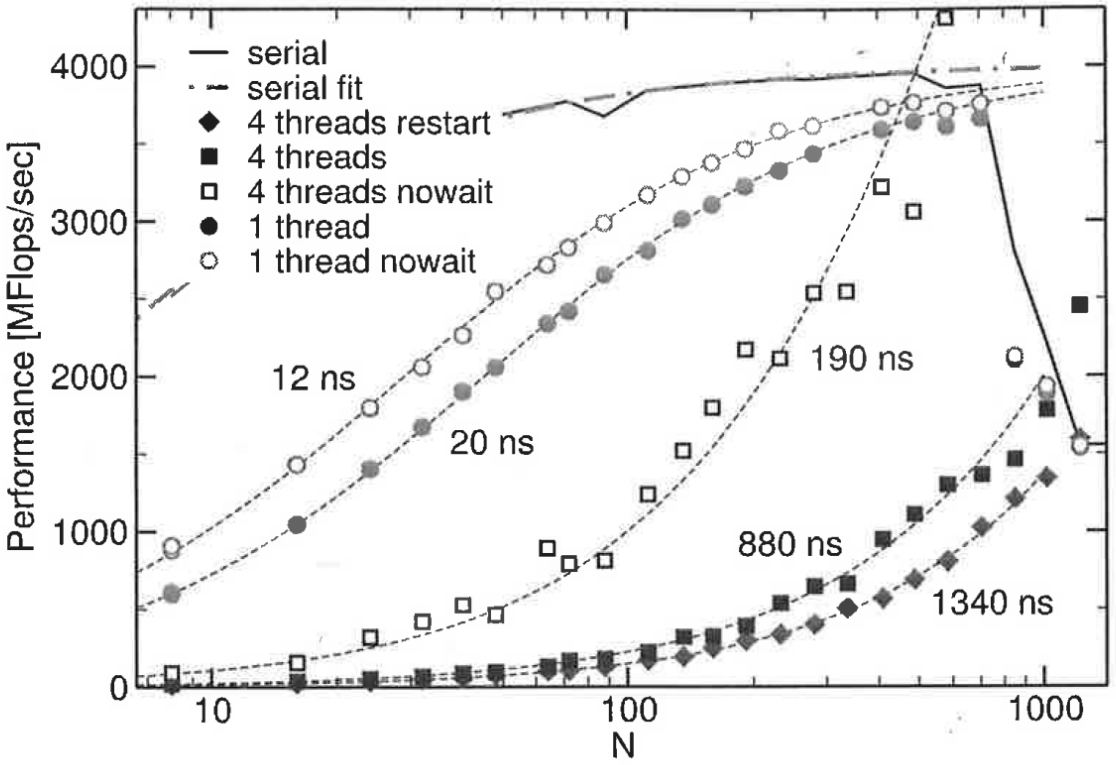
To get

$$P(N,nt) = \frac{1}{(ntP_{\max})^{(-1)} + (T_0 + T_{setup}(nt))/(2\,\text{N})}$$

OpenMP
Loop overhead

Fit model on previous slide to data. Note Barrier impact removed by adding nowait

Diamonds show what happens When threads restarted on every iteration




```
for (j = 1, j < niter, j++)
{ # pragma omp parallel for schedule(static) nowait
   for(i=1, i<N ,i++)
     {
       a(i) = b(i)+c(i)+d(i);
     }
 }
```

OpenMP Loop overhead Conclusions

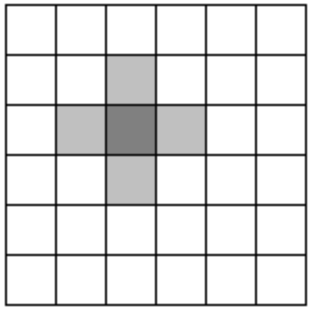
Measurable O/H from running with single OpenMP thread (compiler does not generate truly scalar code)
but for N close to 1000 their performance is similar.

Worksharing loop takes 190ns (570 cycles).

Restarting the team of threads on each iteration takes 460ns or 1380 cycles

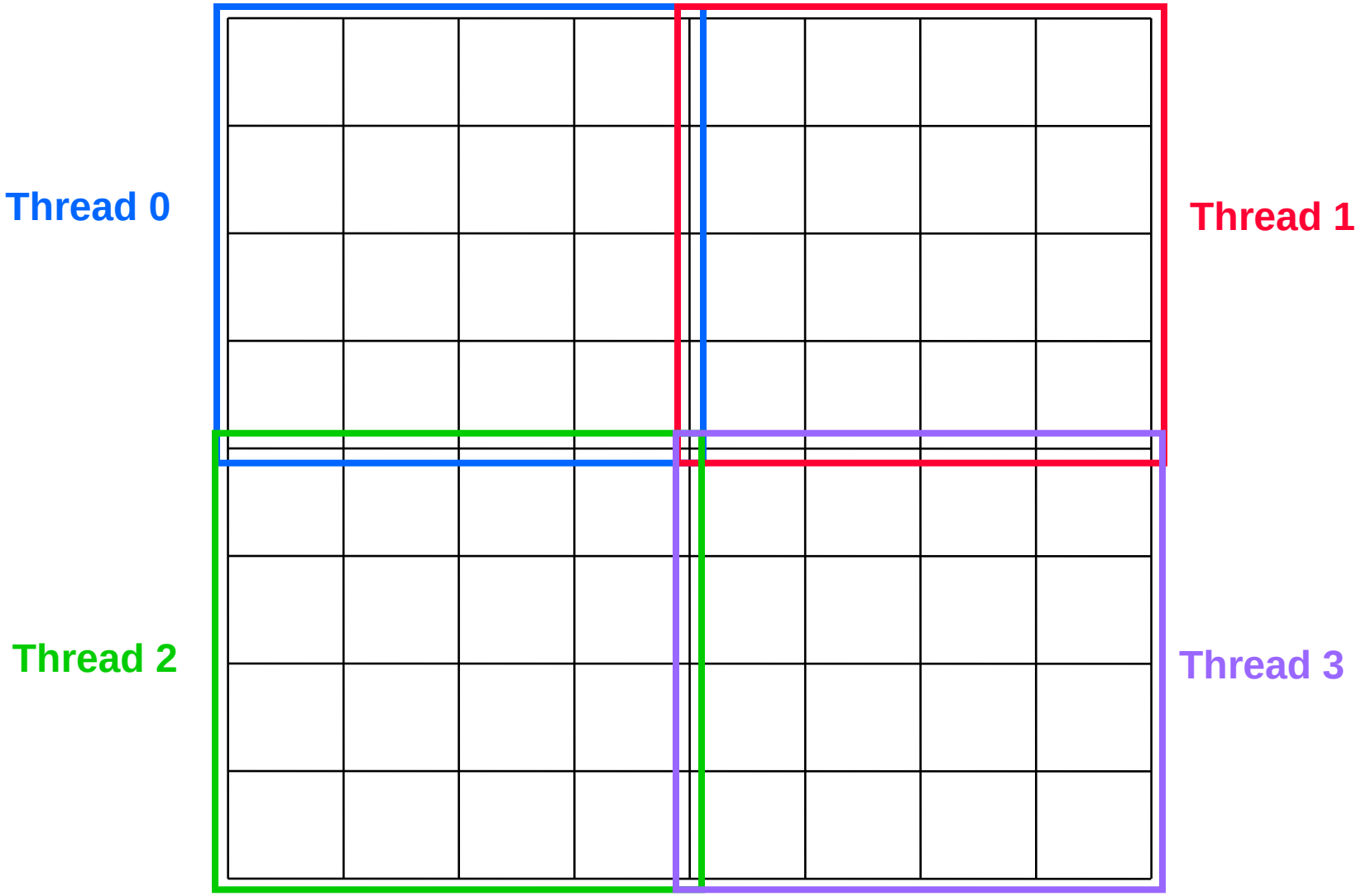
Good idea to minimize number of parallel regions in OpenMP

Microbenchmark: Ocean



- Conceptually similar to SPLASH-2's ocean
- Simulates ocean temperature gradients via succ approximation
  - Operates on a 2D grid of floating point values
- "Embarrassingly" Parallel
  - Each thread operates in a rectangular region
  - Inter-thread communication occurs only on region boundaries
  - Very little synchronization (barrier-only)
- Easy to write in OpenMP!

Microbenchmark: Ocean



Microbenchmark: Grid Relaxation

```
for( t=0; t < t_steps; t++) {
    #pragma omp parallel for \
     shared(grid,x_dim,y_dim) private(x,y)
 for( x=0; x < x_dim; x++) {
     for( y=0; y < y_dim; y++) {
       grid[x][y] = /* avg of neighbors */
     }
  } // Implicit Barrier Synchronization
    temp_grid = grid;
    grid = other_grid;
    other_grid = temp_grid;
}
```

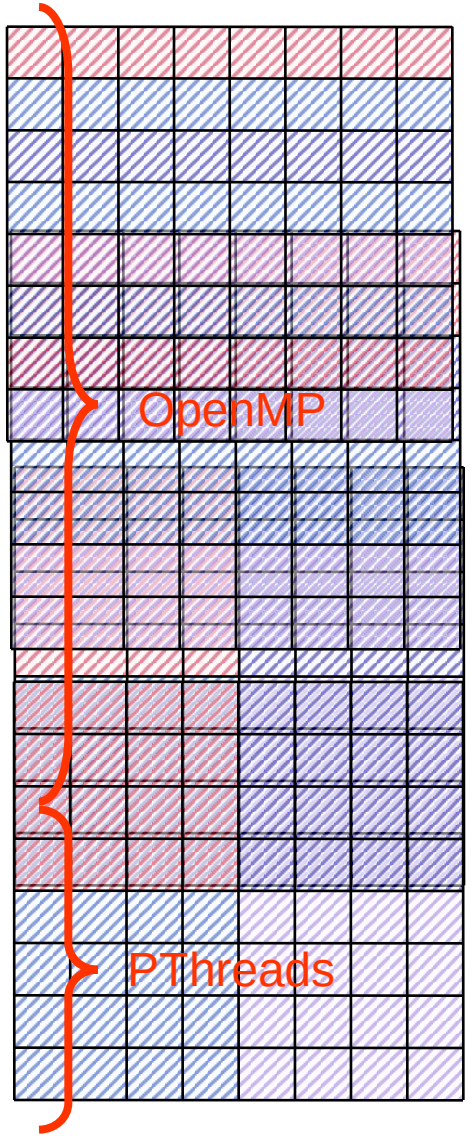
Microbenchmark: Structured Grid

- **`ocean_dynamic`** – Traverses entire ocean, row-by-row, assigning row iterations to threads with `dynamic` scheduling.

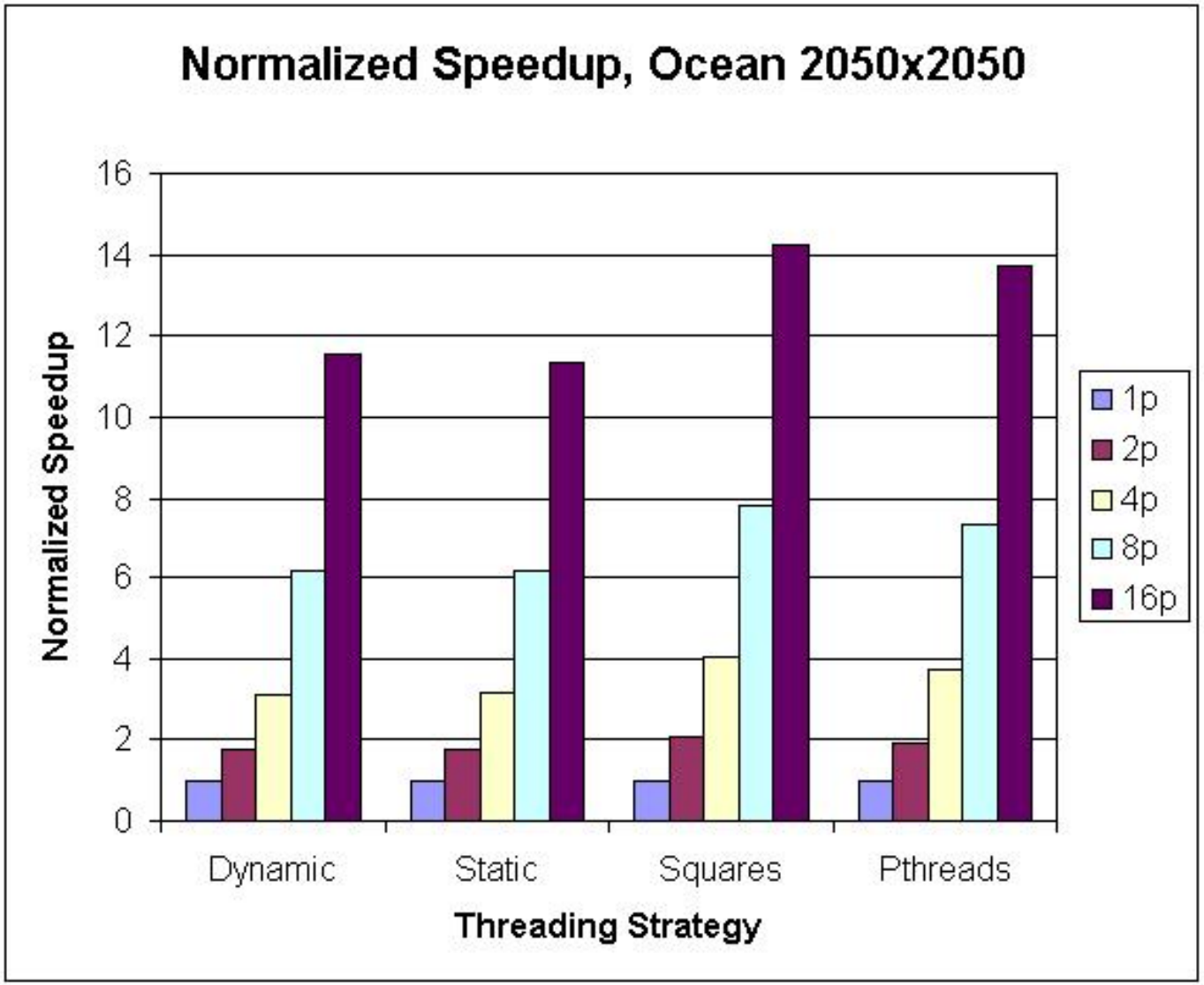
- **`ocean_static`** – Traverses entire ocean, row-by-row, assigning row iterations to threads with `static` scheduling.

- **`ocean_squares`** – Each thread traverses a square-shaped section of the ocean. Loop-level scheduling not used—loop bounds for each thread are determined explicitly.

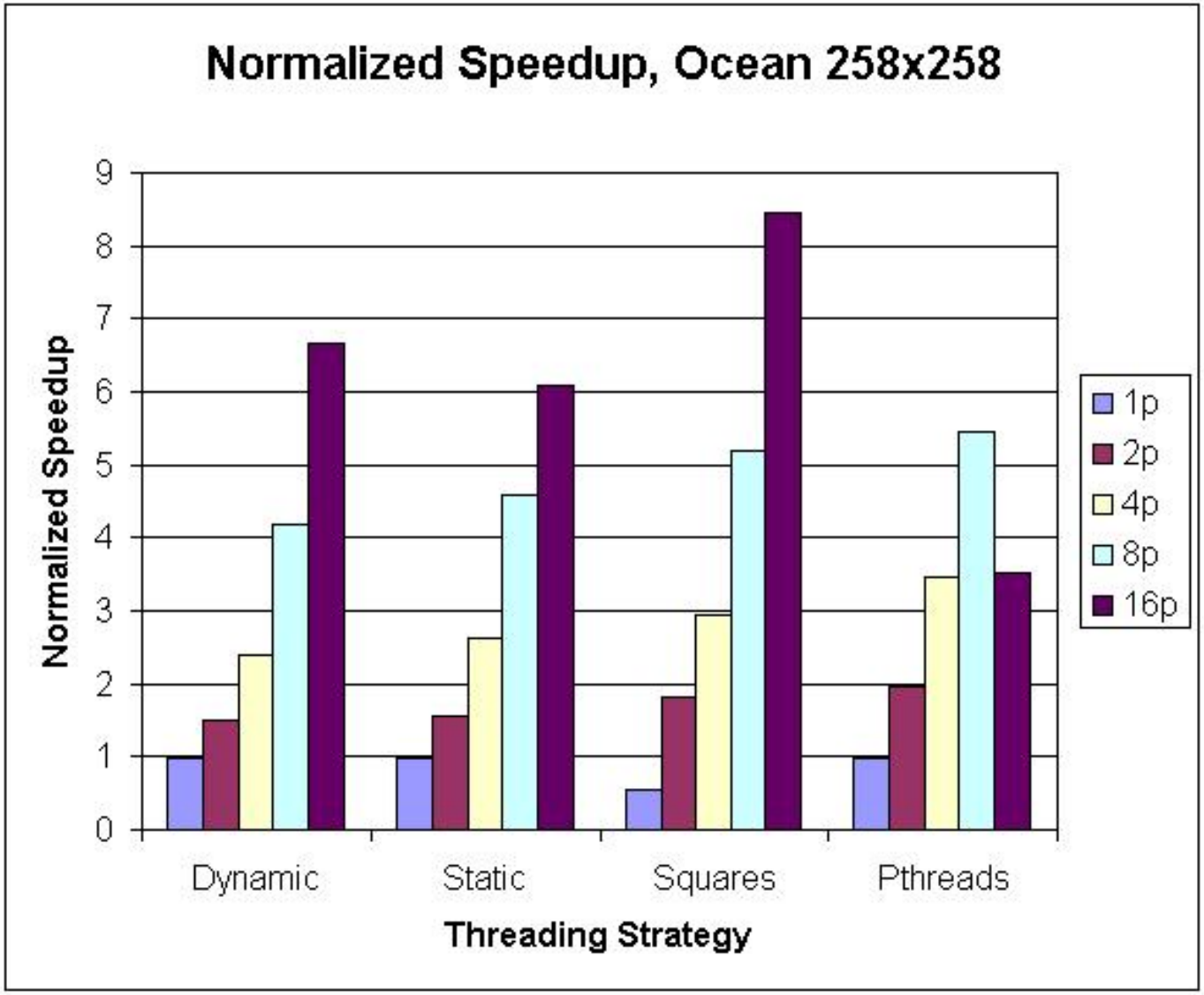
- **`ocean_pthreads`** – Each thread traverses a square-shaped section of the ocean. Loop bounds for each thread are determined explicitly.



Microbenchmark: Ocean



Microbenchmark: Ocean



Evaluation

- OpenMP scales to 16-processor systems
  - Was overhead too high?
    - In some cases, yes
  - Did compiler-generated code compare to hand-written code?
    - Yes!
  - How did the loop scheduling options affect performance?
    - » `dynamic` or `guided` scheduling helps loops with variable interation runtimes
    - » `static` or predicated scheduling more appropriate for shorter loops

- Is OpenMP the right tool to parallelize scientific application?

SpecOMP (2001)

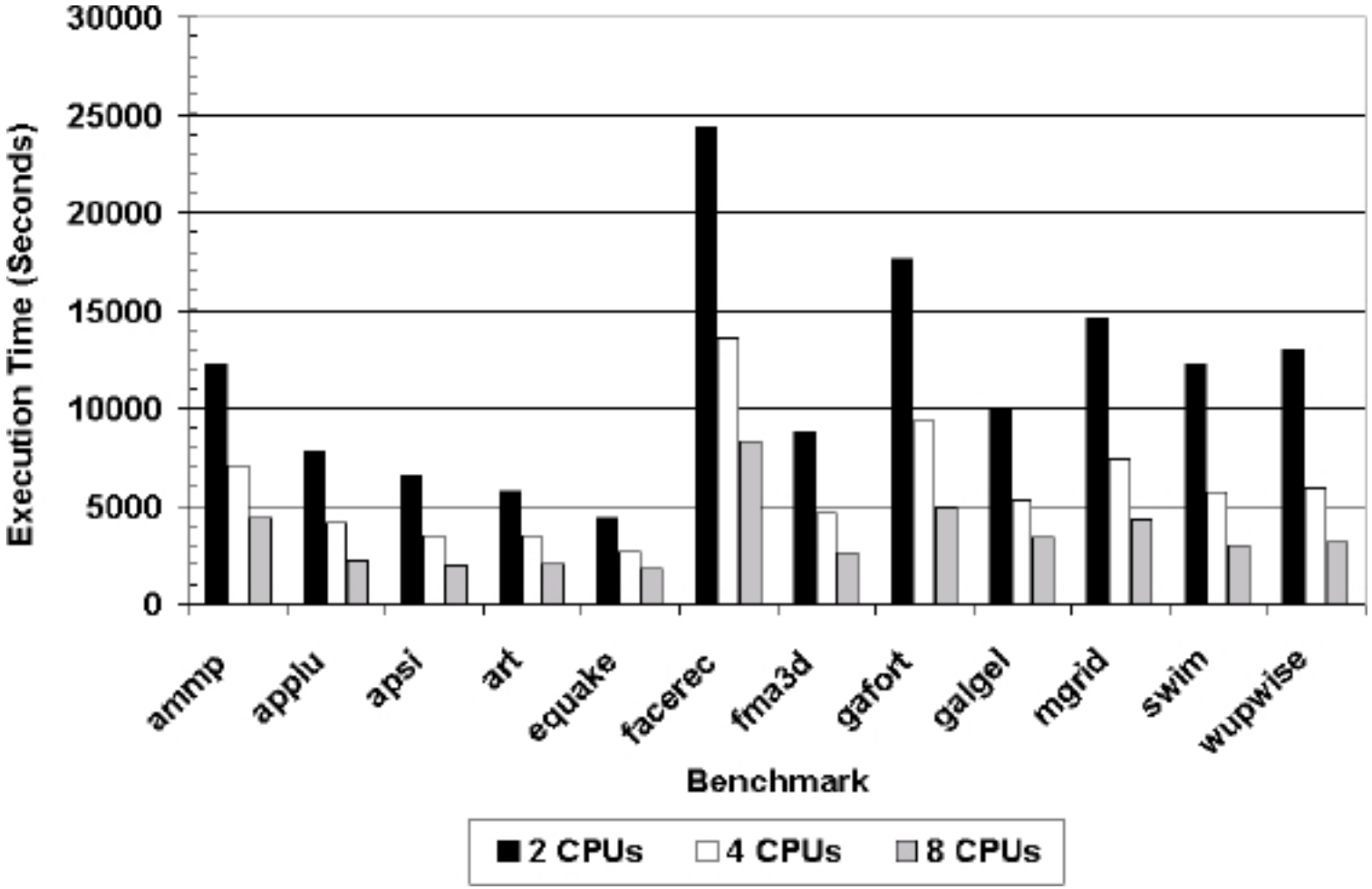
- Parallel form of SPEC FP 2000 using Open MP, larger working sets
  - Aslot et. Al., Workshop on OpenMP Apps. and Tools (2001)

- Many of CFP2000 were “straightforward” to parallelize:
  - *ammp*: 16 Calls to OpenMP API, 13 `#pragmas`, converted linked lists to vector lists
  - *applu*: 50 directives, mostly `parallel` or `do`
  - *fma3d*: 127 lines of OpenMP directives (60k lines total)
  - *mgrid*: automatic translation to OpenMP
  - *swim*: 8 loops parallelized

SpecOMP

Benchmark	Lines of Code	Parallel Cov.	# Par. Sections
ammp	13,500 (C)	99.2 %	7
applu	4,000	99.9 %	22
apsi	7,500	99.9 %	24
art	1,300 (C)	99.5 %	3
facerec	2,400	99.9 %	2
fma3d	60,000	99.5 %	30
gafort	1,500	99.9 %	6
galgel	15,300	96.8 %	29
equake	1,500 (C)	98.4 %	11
mgrid	500	99.9 %	12
swim	400	99.5 %	8
wupwise	2,200	99.8 %	6

SpecOMP - Scalability



Aslot et. Al. Execution times on a “generic” 350Mhz machine.

Overhead of OpenMP Commands (i)

- Most synchronization barriers tens of thousands of cycles
- Barrier 27K TO 30K cycles on Intel, 7K to 31k on IBM
- Loop construct 28K to 40K on Intel OpenMP or 2K to 100K cycles on IBM OpenMP.

Source CLOMP paper [Bronevetsky,Gyllenhall,deSupinski]

Overhead of OpenMP Commands (ii)

Construct	Cost micro sec	Scalability
Parallel	1.5	Linear
Barrier	1.0	Linear or log(n)
Schedule(static)	1.0	Linear
Schedule(guided)	6.0	Depends on contention
Schedule(dynamic)	50	Depends on contention
ordered	0.5	Depends on contention
single	1.0	Depends on contention
Reduction	2.5	Linear or log(n)
Atomic	0.5	Depends on datatype + H/W
Critical	0.5	Depends on contention
Lock/unlock	0.5	Depends on contention

Source Multi Core Programming [Akhter+Roberts]

Summary

- Performance of OpenMP is clearly an issue
- The ease of programming is counterbalanced by uncertain performance
- OpenMP has its advocates and its detractors
- An efficient OpenMP like structure has an important role to play wrt the multicore part of a large parallel machine