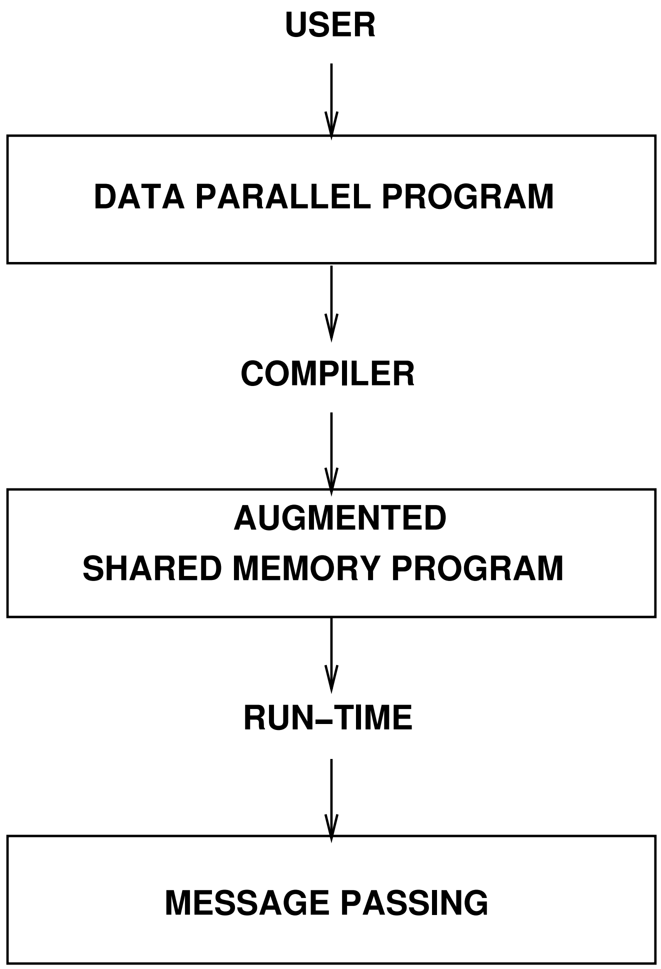
DSM as a Parallelizing Compiler Target



Why Compile to DSM?

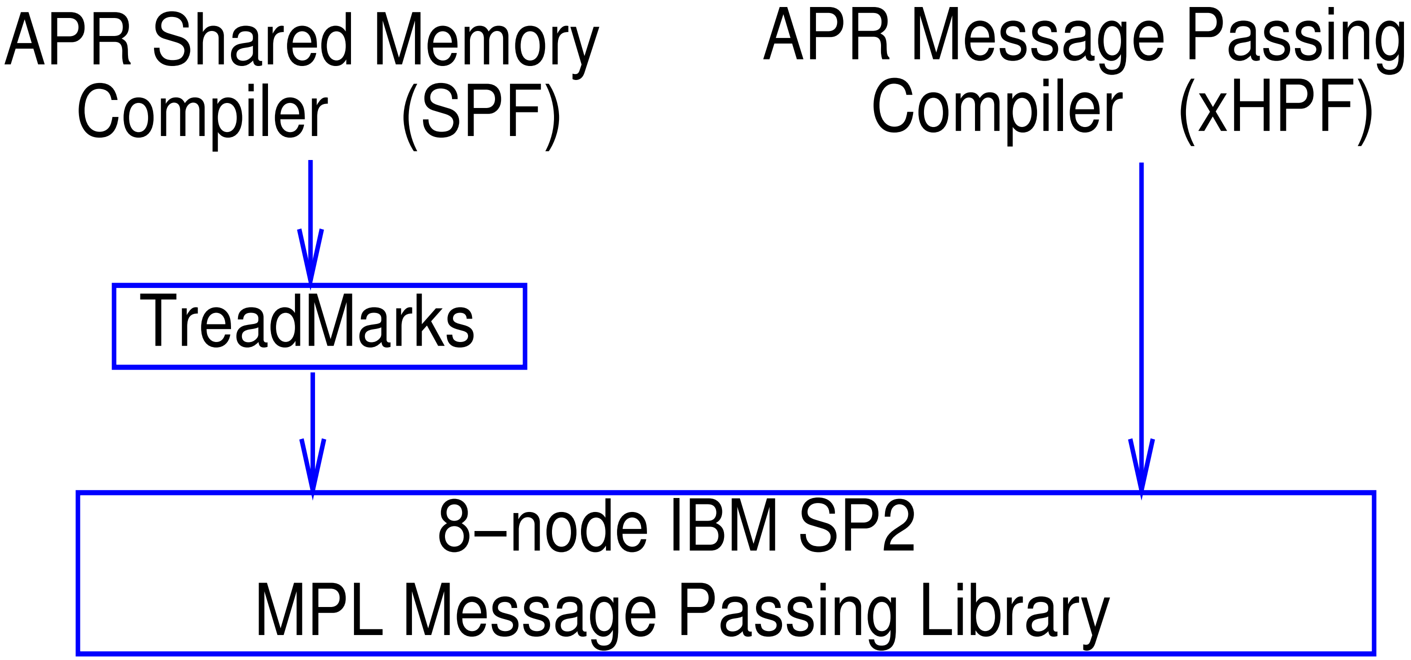
Easier to generate parallel codes

Assume a loop is marked as parallelizable:

- Message passing
  - Compute what to send and where to send
- DSM
  - Add barriers before and after the loop

But, DSM may be slower than message passing

Overview of Experiments



2 irregular applications and 4 regular applications

Shared Memory vs. Message Passing Compilers

	Shared Memory	Message Passing
Annotation	DoPar	DoPar
Data Partition	No	Yes
Model	Fork-join	SPMD
Sequential Part	Master	All

Regular Programs

Jacobi – 2048×2048

- 5-point stencil

Shallow – 1024×1024

- 13 equal-sized 2-D arrays in wrap-around format

MGS – 1024×1024

- Compute an orthonormal basis

3-D FFT – $128 \times 128 \times 64$

- Apply 1-D FFT to all the vectors

Irregular Programs

Irregular

– access pattern cannot be recognized at compile time

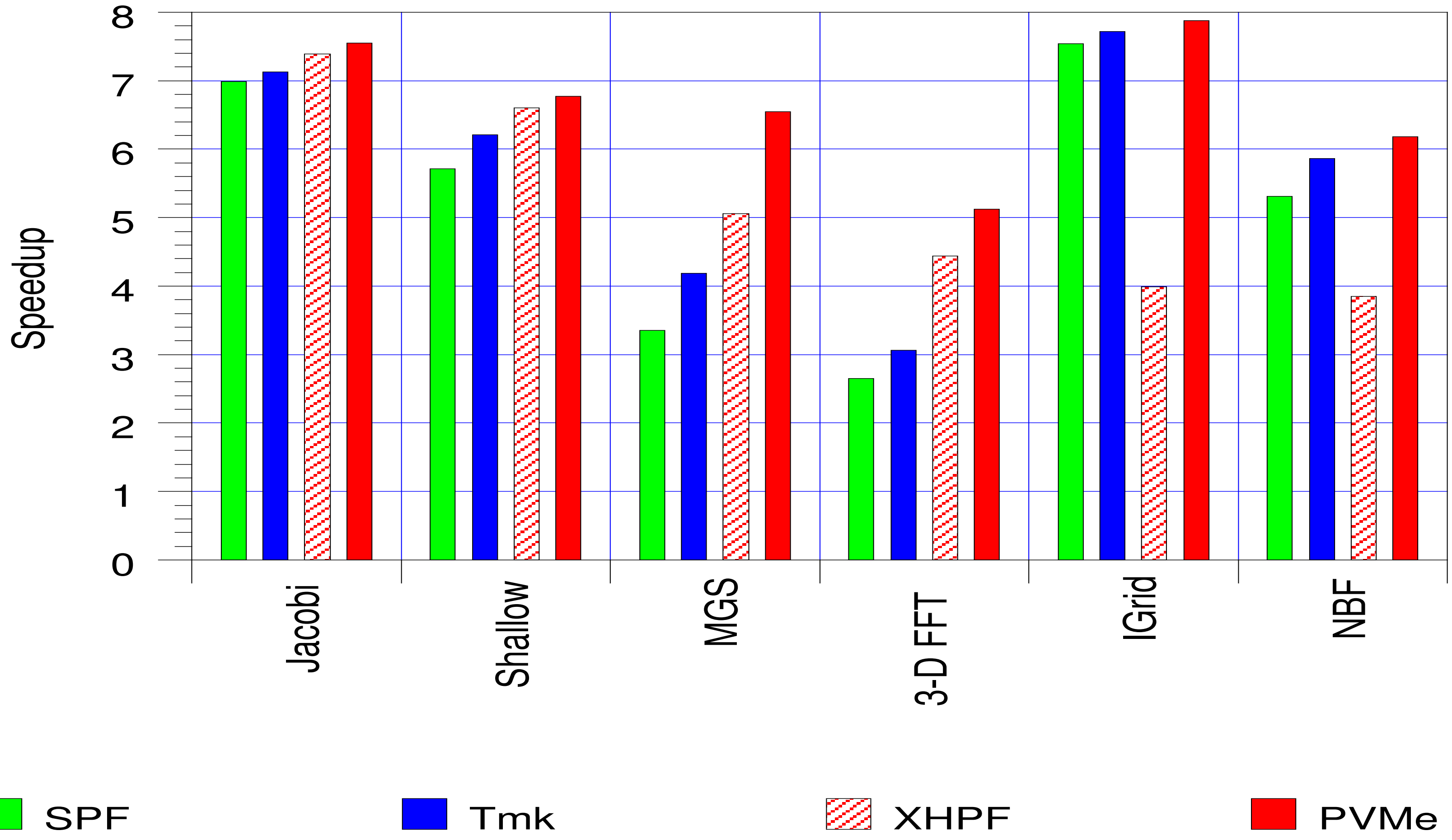
IGrid – 500×500

- 9-point stencil
- List of neighbors

NBF – 32768 molecules

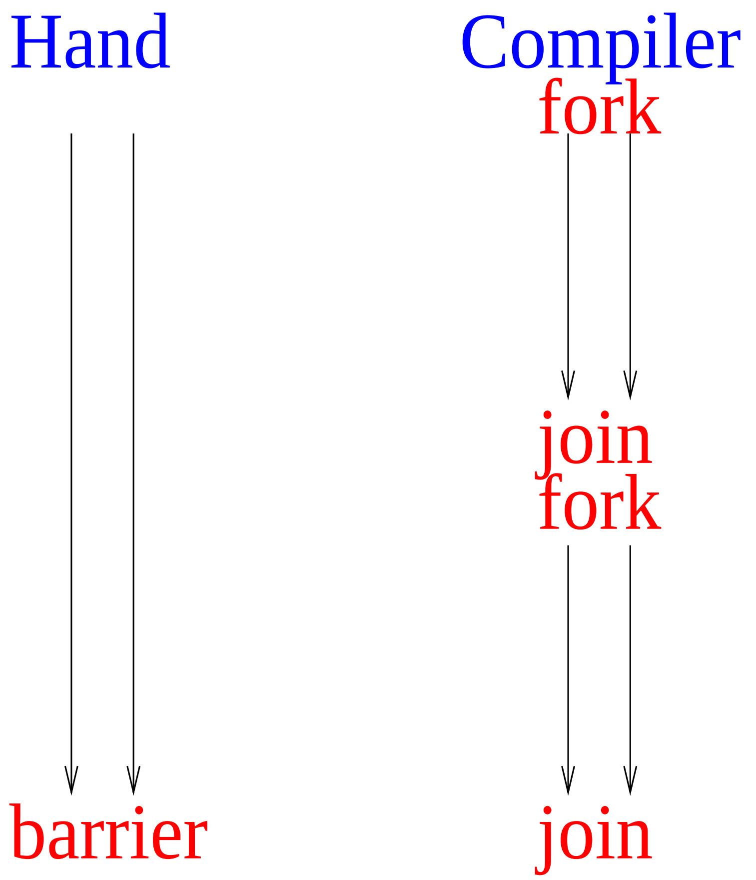
- Molecular dynamics simulation
- List of interacting molecules

SPF Versus XHPF – 8-Processor Speedup

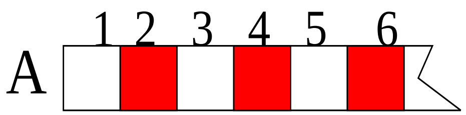


Compiler Induced Overhead – Excess Synchronization

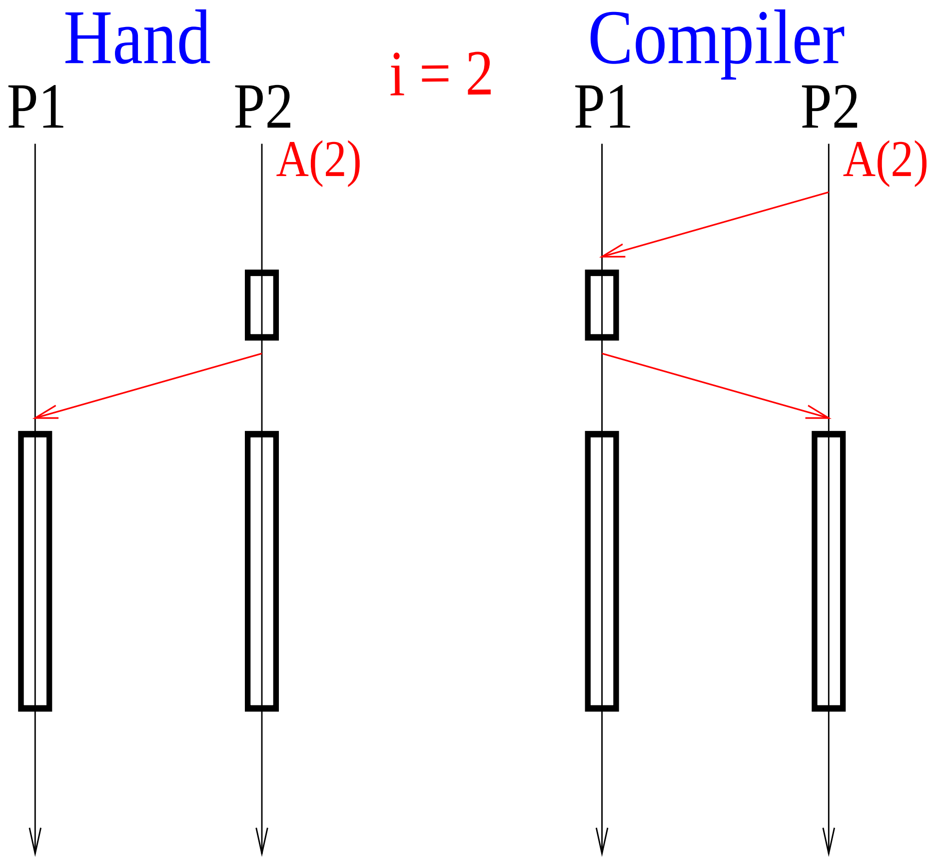
```
par do k = 1, n
      do j = 1, n
         do i = 1, n
            A(i,j,k) = ...
    enddo

par do k = 1, n
      do i = 1, n
         do j = 1, n
            A(i,j,k) = ...
    enddo
```



Compiler Induced Overhead – Poor Processor Locality




```
do i = 1, n
   A(i) = 1/A(i)
par do j = i+1, n
      A(j) = A(j)*A(i)
   enddo
enddo
```



Summary

Why is DSM faster for irregular apps?

- Compiler relies on DSM to get data on demand

Why is DSM slower for regular apps?

- DSM overhead
  - Lack of aggregation
- Compiler induced overhead
  - Excess synchronization
  - Lack of processor locality