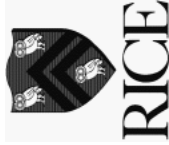


OpenMP in Practice

Charles Koebel
Rice University



NPACI: National Partnership for Advanced Computational Infrastructure

Outline

- ¶ Introduction
- ➔ Basic Parallel Features
- Designing Programs in OpenMP

OpenMP in Practice

2

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Outline

- ¶ Introduction
- ➔ Basic Parallel Features
- Designing New Programs in OpenMP

OpenMP in Practice

3

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

OpenMP

- A portable fork-join parallel model for shared-memory architectures
- Portable
 - Based on Parallel Computing Forum (PCF)
 - Fortran 77 binding here today; C coming this year
- Fork-join model
 - Execution starts with one thread of control
 - Parallel regions fork off new threads on entry
 - Threads join back together at the end of the region
- Shared memory
 - (Some) Memory can be accessed by all threads

OpenMP in Practice

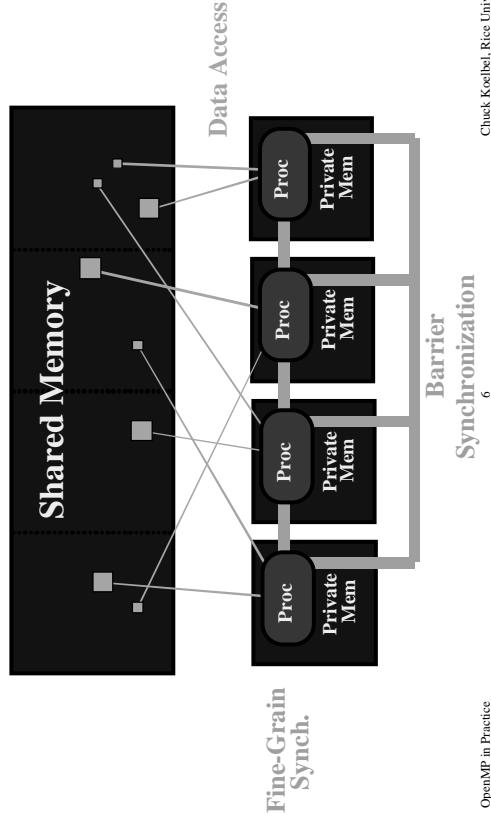
Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

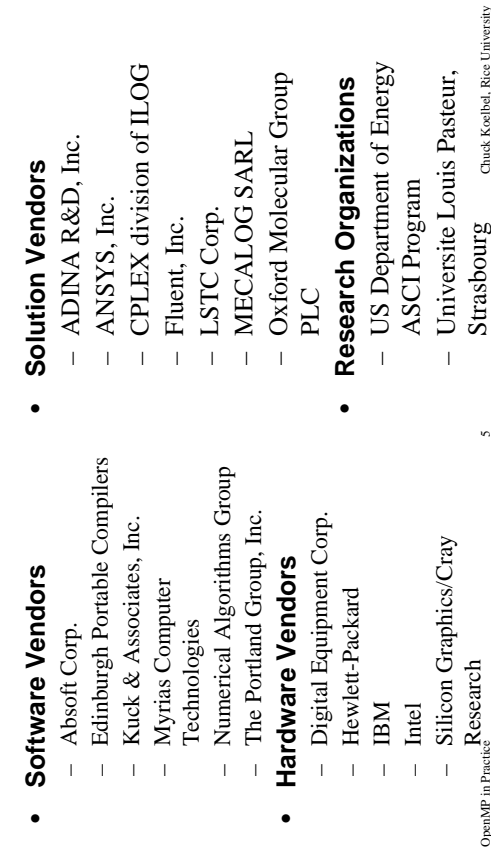
Who's In OpenMP?

- **Software Vendors**
 - Absoft Corp.
 - Edinburgh Portable Compilers
 - Kuck & Associates, Inc.
 - Myrias Computer Technologies
 - Numerical Algorithms Group
 - The Portland Group, Inc.
- **Hardware Vendors**
 - Digital Equipment Corp.
 - Hewlett-Packard
 - IBM
 - Intel
 - Silicon Graphics/Cray Research
- **Solution Vendors**
 - ADINA R&D, Inc.
 - ANSYS, Inc.
 - CPLEX division of ILOG
 - Fluent, Inc.
 - LSTC Corp.
 - MECALOG SARL
 - Oxford Molecular Group PLC
- **Research Organizations**
 - US Department of Energy ASCI Program
 - Universite Louis Pasteur, Strasbourg

Shared Memory in Pictures



OpenMP in Pictures



Outline

- ✓ Introduction
- Basic Parallel Features
 - ★ Control structures
 - ★ Data environment
 - ★ Synchronization
 - ★ Run-time environment and library
- Designing Programs in OpenMP

Design of OpenMP

- “A flexible standard, easily implemented across different platforms”
- Control structures
 - Minimal for simplicity and encouraging common cases
 - PARALLEL, DO, SECTIONS, SINGLE, MASTER
- Data environment
 - New data access capabilities for forked threads
 - SHARED, PRIVATE, REDUCTION

OpenMP in Practice

9

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Design of OpenMP (2)

- Synchronization
 - Simple implicit synch at beginning and end of control structures
 - Explicit synch for more complex patterns: BARRIER, CRITICAL, ATOMIC, FLUSH, ORDERED
- Runtime environment and library
 - Manages modes for forking and scheduling threads
 - E.g, OMP_GET_THREAD_NUM

OpenMP in Practice

10

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Design of OpenMP

- “A flexible standard, easily implemented across different platforms”
- Control structures
 - Minimal for simplicity and encouraging common cases
 - PARALLEL, DO, SECTIONS, SINGLE, MASTER
- Data environment
 - New data access capabilities for forked threads
 - SHARED, PRIVATE, REDUCTION

OpenMP in Practice

9

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Design of OpenMP (2)

- Synchronization
 - Simple implicit synch at beginning and end of control structures
 - Explicit synch for more complex patterns: BARRIER, CRITICAL, ATOMIC, FLUSH, ORDERED
- Runtime environment and library
 - Manages modes for forking and scheduling threads
 - E.g, OMP_GET_THREAD_NUM

OpenMP in Practice

10

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Design of OpenMP

- “A flexible standard, easily implemented across different platforms”
- Control structures
 - Minimal for simplicity and encouraging common cases
 - PARALLEL, DO, SECTIONS, SINGLE, MASTER
- Data environment
 - New data access capabilities for forked threads
 - SHARED, PRIVATE, REDUCTION

OpenMP in Practice

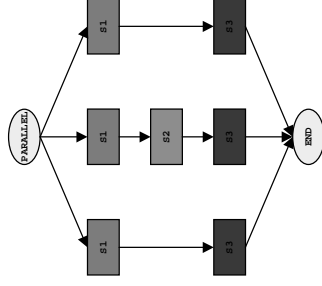
9

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Control Structures

- PARALLEL / END PARALLEL
 - The actual fork and join
 - Number of threads won't change inside parallel region
 - Single Program Multiple Data (SPMD) execution within region
- SINGLE / END SINGLE
 - (Short) sequential section
- MASTER / END MASTER
 - SINGLE on master processor



```
!$OMP PARALLEL
CALL S1
!$OMP SINGLE
CALL S2
!$OMP END SINGLE
CALL S3
!$OMP END PARALLEL
```

OpenMP in Practice

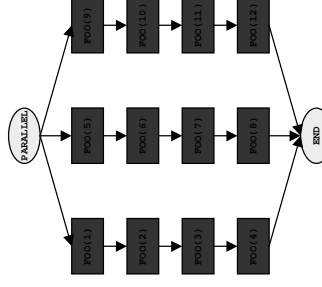
11

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Control Structures (2)

- DO / END DO
 - The classic parallel loop
 - Inside parallel region
 - Or convenient combined directive: **PARALLEL DO**
 - Iteration space is divided among available threads
 - More on how later
 - Loop index is private to thread by default
 - More on other variables later



```
!$OMP PARALLEL
!$OMP DO
DO J = 1, 12
CALL FOO(J)
END DO
!$OMP END DO
```

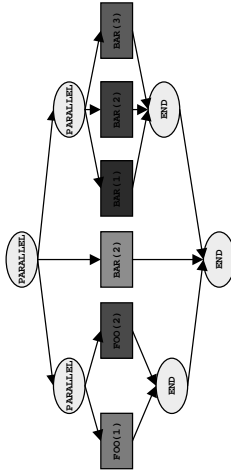
OpenMP in Practice

12

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Control Structures (3)



- **SECTIONS / END SECTIONS**
 - Task parallelism, potentially MIMD
 - **SECTION** marks tasks
 - Inside parallel region
- **Nested parallelism**
 - Requires creating new parallel region
 - Not supported on all OpenMP implementations
 - If no allowed, inner **PARALLEL** is a no-op

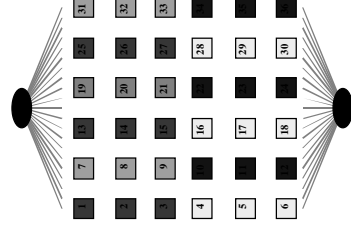
```

!$OMP PARALLEL SECTIONS
!$OMP SECTION
!$OMP PARALLEL DO
DO J = 1, 2
CALL FOO(J)
END DO
!$OMP END SECTION
CALL BAR(2)
!$OMP SECTION
!$OMP PARALLEL DO
DO K = 1, 3
CALL BAZ(K)
END DO
!$OMP END DO
!$OMP END PARALLEL
    
```

13

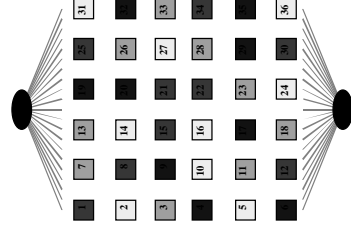
OpenMP in Practice
Chuck Koebel, Rice University

DO Scheduling



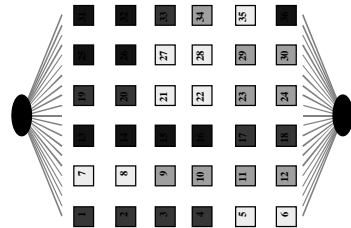
```

!$OMP PARALLEL DO &
!$OMP SCHEDULE(STATIC,3)
DO J = 1,36
CALL SUBR(J)
END DO
!$OMP END DO
    
```



```

!$OMP PARALLEL DO &
!$OMP SCHEDULE(DYNAMIC,1)
DO J = 1,36
CALL SUBR(J)
END DO
!$OMP END DO
    
```



```

!$OMP PARALLEL DO &
!$OMP SCHEDULE(GUIDED,1)
DO J = 1,36
CALL SUBR(J)
END DO
!$OMP END DO
    
```

OpenMP in Practice

14
Chuck Koebel, Rice University

Orphaned Directives

```

PROGRAM main
!$OMP PARALLEL
CALL foo()
CALL bar()
CALL error()
!$OMP END PARALLEL

SUBROUTINE foo()
!$OMP DO
DO i = 1, n
...
END DO
!$OMP END DO
END

SUBROUTINE bar()
!$OMP SECTIONS
!$OMP SECTION
CALL section1()
!$OMP SECTION
...
!$OMP SECTION
...
!$OMP END SECTIONS
END
    
```

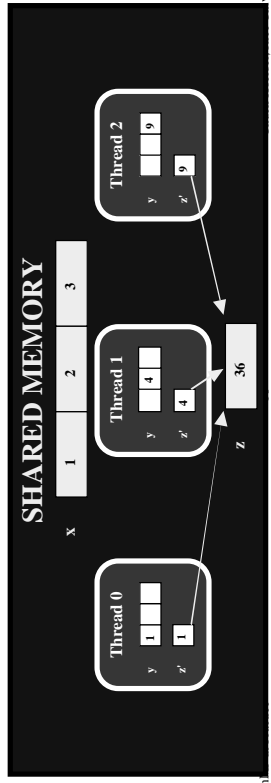
15

OpenMP in Practice
Chuck Koebel, Rice University

OpenMP Data Environments

```

INTEGER x(3), y(3), z
!$OMP PARALLEL DO DEFAULT(PRIVATE), SHARED(x), &
!$OMP REDUCTION(+:z)
DO k = 1, 3
x(k) = k
y(k) = k*k
z = z + x(i)*y(i)
END DO
!$OMP END PARALLEL DO
    
```



OpenMP

Chuck Koebel, Rice University

OpenMP Synchronization

- **Implicit barriers wait for all threads**
 - DO, END DO
 - SECTIONS, END SECTIONS
 - SINGLE, END SINGLE
 - MASTER, END MASTER
 - NOWAIT at END can override synch
 - Global barriers \Rightarrow all threads must hit in the same order

OpenMP Synchronization (2)

- **Explicit directives provide finer control**
 - BARRIER — must be hit by all threads in team
 - CRITICAL (*name*), END CRITICAL — Only one thread may enter at a time
 - ATOMIC — Single-statement critical section for reduction
 - FLUSH (*list*) — “Synchronization point at which the implementation is required to provide a consistent view of memory”
 - ORDERED — For pipelining loop iterations

OpenMP Environment & Runtime Library

- **For controlling execution**
 - Needed for tuning, but may limit portability
 - Control through environment variables or runtime library calls
 - Runtime library takes precedence in conflict

OpenMP Environment & Runtime (2)

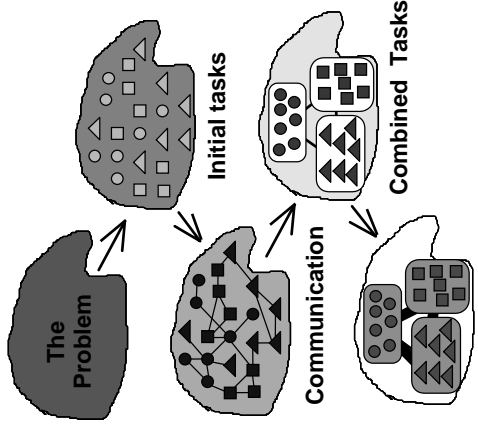
- **OMP_NUM_THREADS: How many to use in parallel region**
 - OMP_GET_NUM_THREADS, OMP_SET_NUM_THREADS
 - Related: OMP_GET_THREAD_NUM, OMP_GET_MAX_THREADS, OMP_GET_NUM_PROCS
- **OMP_DYNAMIC: Should runtime system choose number of threads?**
 - OMP_GET_DYNAMIC, OMP_SET_DYNAMIC
- **OMP_NESTED: Should nested parallel regions be supported?**
 - OMP_GET_NESTED, OMP_SET_NESTED
- **OMP_SCHEDULE: Choose DO scheduling option**
 - Used by RUNTIME clause
- **OMP_IN_PARALLEL: Is the program in a parallel region?**

Outline

- ✓ Introduction
- ✓ Loop and Control Parallelism
- ☞ Designing Programs in OpenMP
 - ★ Basics
 - ★ Irregular Mesh example
 - ★ Optimization issues

Designing Parallel Programs in OpenMP

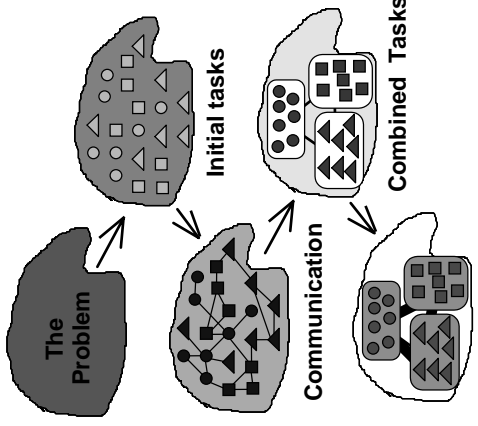
- **Partition**
 - Divide problem into tasks
- **Communicate**
 - Determine amount and pattern of communication
- **Agglomerate**
 - Combine tasks
- **Map**
 - Assign agglomerated tasks to physical processors



Final Program

Designing Parallel Programs in OpenMP

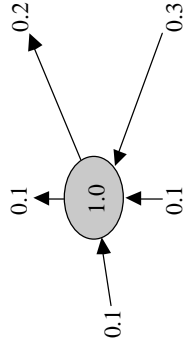
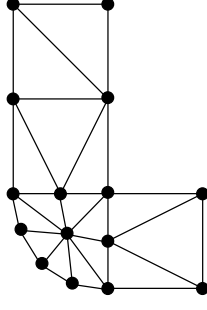
- **Partition**
 - In OpenMP, look for any independent operations (loop parallel, task parallel)
- **Communicate**
 - In OpenMP, look for synchrony points and dependences
- **Agglomerate**
 - In OpenMP, mark parallel loops an/or parallel sections
- **Map**
 - In OpenMP, implicit or explicit scheduling
 - Data mapping goes outside the standard



Final Program

Irregular Mesh: The Problem

- **The Problem**
 - Given an irregular mesh of values
 - Update each value using its neighbors in the mesh
- **The Approach**
 - Store the mesh as a list of edges
 - Process all edges in parallel
 - Compute contribution of edge
 - Add to one endpoint, subtract from the other



Irregular Mesh: Sequential Program

```

REAL x(nnode), Y(nnode), flux
INTEGER iedge(nedge,2)
err = tol * 1e6
DO WHILE (err > tol)
  DO i = 1, nedge
    flux = (Y(iedge(i,1))-Y(iedge(i,2))) / 2
    x(iedge(i,1)) = x(iedge(i,1)) - flux
    x(iedge(i,2)) = x(iedge(i,2)) + flux
    err = err + flux(i)*flux(i)
  END DO
  err = err / nedge
  DO j = 1, nnode
    Y(i) = x(i)
  END DO
END DO

```

OpenMP in Practice

25

Chuck Koebel, Rice University

NPACI, National Partnership for Advanced Computational Infrastructure

Irregular Mesh: OpenMP Partitioning

- Flux computations are data-parallel
 - $\text{flux} = (\text{x}(\text{iedge}(i,1)) - \text{x}(\text{iedge}(i,2))) / 2$
- Node updates are nearly data-parallel
 - $\text{x}(\text{iedge}(i,1)) = \text{x}(\text{iedge}(i,1)) - \text{flux}$
 - Not independent if $\text{iedge}(i_y,1) = \text{iedge}(i_x,2)$
 - But ATOMIC supports associative updates
- Error check is a reduction
 - $\text{err} = \text{err} + \text{flux}(i) * \text{flux}(i)$
 - REDUCTION class for variables

OpenMP in Practice

26

Chuck Koebel, Rice University

NPACI, National Partnership for Advanced Computational Infrastructure

Irregular Mesh: OpenMP Communication & Agglomeration

- Communication needed for all parts
 - Edge-node (computing flux); node-node (computing \mathbf{x}); reduction (computing err)
 - Communication handled simply by shared or reduction variables
- Because of the tight ties between flux , \mathbf{x} , and err , keep the loop intact
 - Incremental parallelization via OpenMP works perfectly
 - Any agglomeration scheme balances computation load
 - Agglomeration will change locality, though

OpenMP in Practice

27

Chuck Koebel, Rice University

NPACI, National Partnership for Advanced Computational Infrastructure

Irregular Mesh: OpenMP Program

```

!$OMP PARALLEL, DEFAULT(SHARED)
!$OMP SINGLE
err = tol * 1e6
!$OMP END SINGLE
!$OMP END SINGLE
DO WHILE (err > tol)
  !$OMP DO, PRIVATE(flux), REDUCTION(+:err)
  DO i = 1, nedge
    flux = (Y(iedge(i,1))-Y(iedge(i,2))) / 2
  !$OMP ATOMIC
    x(iedge(i,1)) = x(iedge(i,1)) - flux
  !$OMP ATOMIC
    x(iedge(i,2)) = x(iedge(i,2)) + flux
    err = err + flux(i)*flux(i)
  END DO
  !$OMP END DO
  !$OMP SINGLE
  err = err / nedge
  !$OMP SINGLE
  !$OMP DO
  DO j = 1, nnode
    Y(i) = x(i)
  END DO
  !$OMP END DO
END DO
!$OMP END PARALLEL

```

OpenMP in Practice

28

Chuck Koebel, Rice University

NPACI, National Partnership for Advanced Computational Infrastructure

Irregular Mesh: OpenMP Mapping

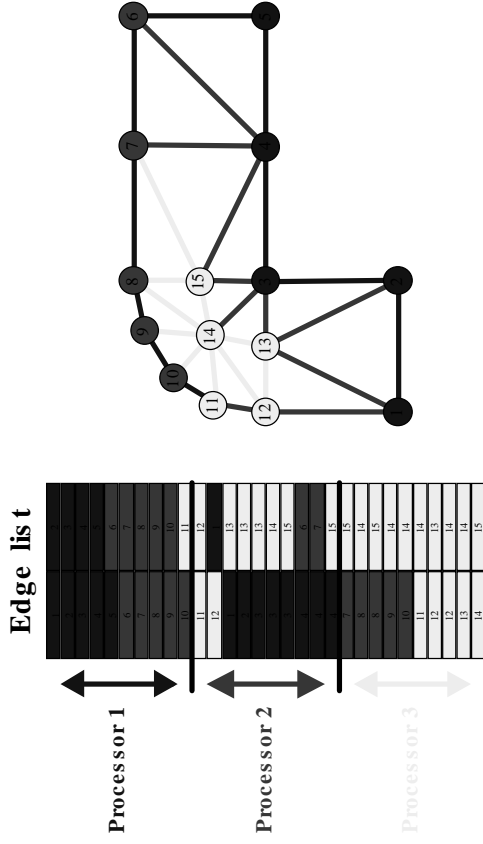
- There may be significant differences in data movement based on scheduling
- The ideal:
 - Every thread runs over its own edges (static scheduling)
 - Endpoints of these edges are not shared
 - Data moves to its home on the first pass, then stays put
- The reality:
 - Connected graph \Rightarrow some endpoints must be shared
 - Multi-word data moves (cache lines) \Rightarrow false sharing
 - OpenMP does not standardize how to resolve this
 - Best bets: Reorder data for locality or use nonstandard directives (HPF + MPI?)

OpenMP in Practice

Chuck Koelbel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Irregular Mesh: Bad Data Ordering

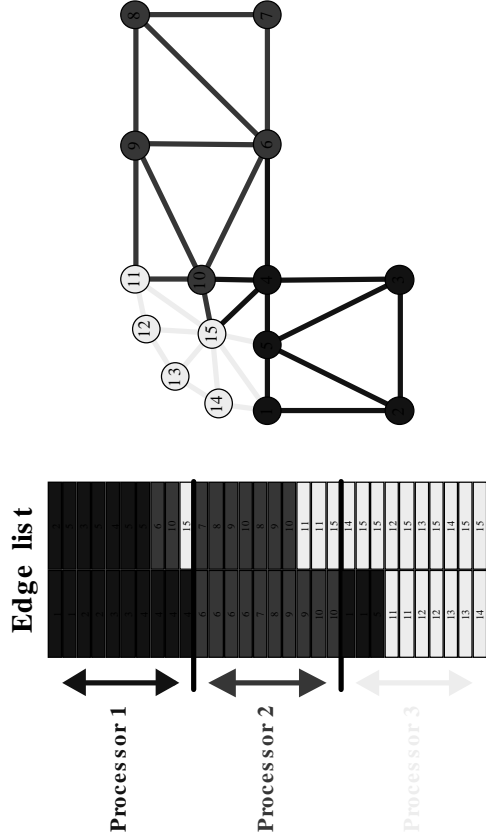


OpenMP in Practice

Chuck Koelbel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Irregular Mesh: Good Data Ordering



OpenMP in Practice

Chuck Koelbel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Irregular Mesh: OpenMP Program with Data Reordering

```

!$OMP PARALLEL, DEFAULT(SHARED)
CALL renumber_nodes( iedge, permute_node )
!$OMP DO
DO i = 1, nnode
  x( permute_node(i) ) = old_x(i)
END DO
!$OMP END DO NOWAIT
!$OMP DO
DO i = 1, nedge
  iedge(i,1) = permute_node(iedge(i,1))
  iedge(i,2) = permute_node(iedge(i,2))
END DO
!$OMP END DO
CALL sort_edges(iedge, nedge)
!$OMP SINGLE
err = tol * le6
!$OMP END SINGLE
DO WHILE (err > tol)
  !$OMP DO, SCHEDULE(STATIC), PRIVATE(err), REDUCTION(+:err)
  ...
  !$OMP END DO
END DO
!$OMP END PARALLEL

```

OpenMP in Practice

Chuck Koelbel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

OpenMP Summary

- **Based on fork-join parallelism in shared memory**
 - Threads start at beginning of parallel region, come back together at end
 - Close to some hardware
 - Linked from traditional languages
- **Very good for sharing data and incremental parallelization**
- **Unclear if it is feasible for distributed memory**
- **For more information:**
 - <http://www.openmp.org>

OpenMP in Practice

33

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

Optional Topic: Comparing and combining systems

NPACI: National Partnership for Advanced Computational Infrastructure

Three Systems Compared

- **HPF**
 - Good abstraction: data parallelism
 - System hides many details from programmer
 - Well-suited to regular problems & machines w/ locality
- **MPI**
 - Lower-level abstraction: message passing
 - System works everywhere, including new systems
 - Well-suited to distributed memory, but requires work
- **OpenMP**
 - Good abstraction: fork-join
 - Incremental parallelization on shared memory
 - No implementations yet on distributed memory
 - Well-suited for any application if locality is not an issue

OpenMP in Practice

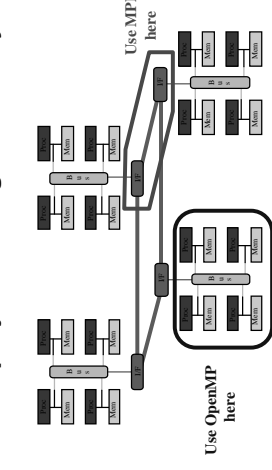
35

Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

OpenMP + MPI

- **Modern parallel machines are often shared memory nodes connected by message passing**
- **Can be programmed by calling MPI from OpenMP**
 - MPI implementation must be thread-safe
- **ASCI project is using this heavily**



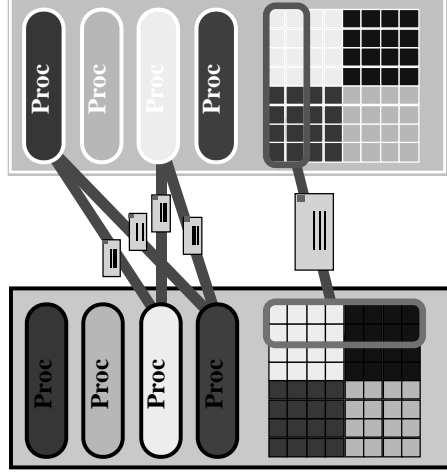
OpenMP in Practice 36

SGI Origin2000
Chuck Koebel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

MPI + HPF

- Many applications consist of several data-parallel modules
- Can link HPF codes on different machines using MPI
 - Requires special MPI implementation and runtime
- HPFMPI project at Argonne has done proof-of-concept



OpenMP in Practice

37

Chuck Koelbel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure

HPF + OpenMP

- HPF can be implemented by translating it to OpenMP
 - Good idea on shared-memory machines
 - May have real advantages for optimizing locality and data layout
- HPF may call OpenMP directly
 - Proposal made at HPF Users Group meeting last month
 - Not quite trivial, since HPF and OpenMP may not agree on data layout
 - Things could get worse if MPI is also implemented on OpenMP

OpenMP in Practice

38

Chuck Koelbel, Rice University

NPACI: National Partnership for Advanced Computational Infrastructure