Parallel Software Engineering with OpenMP
Kuck & Associates, Inc.

kai@kai.com, 217-356-2288
http://www.kai.com

Outline
- Introduction
- What is Parallel Software Engineering
- Parallel Software Engineering Issues
- OpenMP
- KAP/Pro for OpenMP
- Conclusions
Why Parallel Software Engineering

We seek the following benefits --

- Performance
- Productivity
- Quality
- Standards

Steps in Parallelizing an Application

- **Analyze**
  - Find the Parallel loop.
  - Make sure it’s the right one.

- **Resructure**
  - Make the necessary modifications:
    - Classify variables, Add synchronization.

- **Test**

- **Improve**
  - Verify that the program basically works in parallel mode.

- **Q/A**
  - Do the tuning necessary to get peak performance.
  - Verify that the parallel application is as robust as serial application.
Digression -- A Bit of History

- Minisupercomputers legacy
  - Sequent, Alliant pioneered in 2nd half of 80’s
- PCF/X3H5 standardization effort
  - Cray, Digital, IBM, SGI developed consensus
- Parallel model used by many companies
- Advances in Shared Memory
  Multiprocessors is causing growing usage

The Trend Towards Portable Parallel Processing

- Portability between systems.
- With a common set of directives, OpenMP
OpenMP

Control Directives

<table>
<thead>
<tr>
<th>Parallel Region</th>
<th>Parallel Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{C$OMP PARALLEL}$</td>
<td>$\text{C$OMP SECTIONS}$</td>
</tr>
<tr>
<td>$\text{C$OMP{IF (if_expression)}}$</td>
<td>$\text{C$OMP SECTION}$</td>
</tr>
<tr>
<td>$\text{C$OMP{SHARED(shared_variables)}}$</td>
<td>$\text{C$OMP END SECTIONS[NOWAIT]}$</td>
</tr>
<tr>
<td>$\text{C$OMP{PRIVATE(local_variables)}}$</td>
<td></td>
</tr>
<tr>
<td>$\text{C$OMP END PARALLEL}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parallel Do</th>
<th>Single Processor Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{C$OMP DO}$</td>
<td>$\text{C$OMP SINGLE}$</td>
</tr>
<tr>
<td>$\text{C$OMP{SCHEDULE(type,chunk)}}$</td>
<td>$\text{C$OMP END SINGLE[NOWAIT]}$</td>
</tr>
<tr>
<td>$\text{C$OMP END DO[NOWAIT]}$</td>
<td></td>
</tr>
</tbody>
</table>

OpenMP Data Directives

<table>
<thead>
<tr>
<th>Parallel Data</th>
<th>Synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{C$OMP THREAD PRIVATE[/common,...]}$</td>
<td>$\text{C$OMP CRITICAL [(variable)]}$</td>
</tr>
<tr>
<td>$\text{C$OMP{COPYIN(variables)}}$</td>
<td>$\text{C$OMP END CRITICAL}$</td>
</tr>
<tr>
<td>$\text{C$OMP{PRIVATE(variables)}}$</td>
<td>$\text{C$OMP ORDERED}$</td>
</tr>
<tr>
<td>$\text{C$OMP{SHARED(variables)}}$</td>
<td>$\text{C$OMP END ORDERED}$</td>
</tr>
<tr>
<td>$\text{C$OMP{FIRSTPRIVATE(variables)}}$</td>
<td>$\text{C$OMP MASTER}$</td>
</tr>
<tr>
<td>$\text{C$OMP{LASTPRIVATE(variables)}}$</td>
<td>$\text{C$OMP END MASTER}$</td>
</tr>
<tr>
<td>$\text{C$OMP{REDUCTION(op : variables)}}$</td>
<td>$\text{C$OMP BARRIER}$</td>
</tr>
<tr>
<td>$\text{C$OMP{DEFAULT}}$</td>
<td>$\text{C$OMP ATOMIC}$</td>
</tr>
<tr>
<td>$\text{C$OMP{PRIVATE[SHARED</td>
<td>NONE]}}$</td>
</tr>
</tbody>
</table>
### OpenMP

#### Library and Environment

<table>
<thead>
<tr>
<th>Run Time Library Routines</th>
<th>Environment Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>external omp_set_num_threads(integer)</code></td>
<td><code>OMP_SCHEDULE</code></td>
</tr>
<tr>
<td><code>integer omp_get_num_threads()</code></td>
<td><code>OMP_NUM_THREADS</code></td>
</tr>
<tr>
<td><code>integer omp_get_max_threads()</code></td>
<td><code>OMP_DYNAMIC</code></td>
</tr>
<tr>
<td><code>integer omp_get_thread_num()</code></td>
<td><code>OMP_NESTED</code></td>
</tr>
<tr>
<td><code>integer omp_get_num_procs()</code></td>
<td></td>
</tr>
<tr>
<td><code>external omp_set_dynamic(logical)</code></td>
<td></td>
</tr>
<tr>
<td><code>logical omp_get_dynamic()</code></td>
<td></td>
</tr>
<tr>
<td><code>logical omp_in_parallel()</code></td>
<td></td>
</tr>
<tr>
<td><code>external omp_set_nested(logical)</code></td>
<td></td>
</tr>
<tr>
<td><code>logical omp_get_nested()</code></td>
<td></td>
</tr>
<tr>
<td><code>external omp_init_lock(var)</code></td>
<td></td>
</tr>
<tr>
<td><code>external omp_init_destroy(var)</code></td>
<td></td>
</tr>
<tr>
<td><code>external omp_set_lock(var)</code></td>
<td></td>
</tr>
<tr>
<td><code>external omp_unset_lock(var)</code></td>
<td></td>
</tr>
<tr>
<td><code>logical omp_test_lock(var)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Parallel Processing Model

1. Parallel Loops
   - **Concept!** Iteration scheduling and barriers
2. Parallel Regions
   - **Concept!** Redundant code execution
3. Private Commons
   - **Concept!** Storage Parallelism
4. Critical Sections, Barriers
   - **Concept!** Structured synchronization
Parallel Loop Model

- Note threads, shared and private variables.

```
program example
  c$omp parallel do
  c$omp& shared(A)
  c$omp& private(I)
  do I=1,100
     A(I) = ...
  end do
  c$omp end parallel do
end
```

Kuck & Associates, Inc.

Parallel Region Model

- Note “redundant” code

```
c$omp parallel
  do j =1,jconverg
    c$omp do
      do i=ilb,iub
        ...
      end do
    end do
  c$omp end parallel
```

Kuck & Associates, Inc.
### Parallel Storage Model

- **Note THREAD PRIVATE**

```
c$omp threadprivate(/A/)
    common /A/ data(100)
c$omp parallel
    c$omp& copyin(/A/)
    ...
    c$omp end parallel

c$omp parallel
    c$omp end parallel
```

Only 3 new areas created on entry. Master keeps original. Values copied in.

Next region inherits values.

---

### Parallel Synchronization Model

- **Critical section and Barrier**

```
c$omp parallel private(i,j)
c$omp& shared(a,b,m,n,sum)
    do 20 i=1,n
        sum = 0
    c$omp do
        do 10 j=1,m
            c$omp critical
                sum=sum+a(j,i)
            c$omp end critical
    10     continue
    c$omp barrier
    b(i) = sum
    20    continue
    c$omp end parallel
```

Mutual exclusion as threads update sum. Wait until all threads have finished column sum.

All threads take turns updating sum.
**OpenMP Feature**

**Dynamic Threads**

**Without Dynamic Threads**

- With Static Threads
  - Request 3 threads
  - Get exactly 3 threads
- With Dynamic Threads
  - Request 3 threads
  - Get 3 threads if available

**With Dynamic Threads**

- Your job gets done faster and your colleagues too
- Avoids over allocating processors

---

**Experience Learned with Directive Programs**

- Ideal for directives
  - Model 1: Fork, share work, join and repeat
- Parallelism in whole program didn’t work
  - Model 2: Fork once - barrier when needed
- OpenMP has added orphaned directives

---

Model 2: **Before** OpenMP

```
c$par parallel
  myid = mpptid()
  ichunk = isize / mppnth()
  mywork = ichunk * myid
  call simul8(myid,mywork,ichunk)
c$par end parallel
... subroutine simul8(myid, mywork, ichunk)
do i=mywork,mywork+ichunk
  call realwork
end do
```

c$par barrier
end
OpenMP Feature
Orphaned Directives

- Now directives don’t have to be in the same subroutine
- Removes need for:
  - Explicit scheduling
  - Passing scheduling arguments
  - Explicit barriers
- Dynamic binding

Model 2: **After** OpenMP

```c
c$omp parallel
call simul8(normal args)
c$omp end parallel
...
subroutine simul8(args)
c$omp do schedule(static)
do i=0,isize
call realwork
end do
end
```

OpenMP IF clause
Reduce parallel overhead

- Optional IF clause for PARALLEL or PARALLEL DO directive
  ```c
  c$omp parallel do if(n .GE. 10*numcpus)
  -- If true, execute region on multiple processors
  -- If false, execute region on single processor
  ```
- Identify short parallel regions that may slow you down
- Select best loop in nested loops at runtime
Example -- Parallel Reduction

- Use private variable to accumulate per thread
- Use critical section in parallel region

```c
%$omp parallel
%$omp& private(i,j,sum_local)
%$omp& shared(a,m,n,sum)
    sum_local = 0.0
%$omp do
    do 10 i=1,n
    do 10 j=1,m
        sum_local=sum_local+a(j,i)
    10    continue
%$omp critical
    sum = sum+sum_local
%$omp end critical
%$omp end parallel
```

Example -- OpenMP Reduction

- OpenMP replaces sum with local_sum, inserts serial reduction to sum
  - Can be scalars or array elements
  - Only simple reductions +, -, *, min, and max

```c
%$omp parallel do
%$omp& private(i,j)
%$omp& shared(a,m,n)
%$omp& reduction(+ : sum)
    do 10 i=1,n
    do 10 j=1,m
        sum=sum+a(j,i)
    10    continue
```
**KPTS + OpenMP = Parallel Software Engineering**

- **Performance** --
  - Meets or beats all modes on scalability
- **Productivity** --
  - Much easier to use than other modes
- **Quality** --
  - Enables parallelism validation for first time
- **Standards** --
  - Defacto becoming fact

---

**OpenMP Parallel Software Engineering with KPTS**

- **Steps in Parallelizing an Application**
  - Analyze
  - Restructure
  - Test
  - Improve
  - Q/A

---

Kuck & Associates, Inc.
What is **KAP**

- **KAP** restructures to **OpenMP** parallelism

  - Analyze
  - Restructure
  - Test
  - Improve
  - Q/A

  Feedback from tuning helps adjust parallelism options.

  KAP used to generate **OpenMP** parallelism.

What is **Guide**

- **Guide** implements **OpenMP** parallelism

  - Analyze
  - Restructure
  - Test
  - Improve
  - Q/A

  User identifies where Parallelism is,

  Guide restructures program to implement **OpenMP**.

  Feedback from tuning helps adjust parallelism options.
What Makes Parallel Debugging Hard?

Think of the things that can go wrong --

- Incorrectly pointing to the same place
- Incorrectly point to different places
- Incorrect initialization of parallel regions
- Not saving values from parallel regions
- Unsynchronized access
- Variation in the execution order

And More ...

More things that can go wrong --

- Inconsistently synchronized I/O statements
- Inconsistent declarations of shared variables
- Parallel stack size problems

Do You Want More?
Tactics For Fixing and Preventing Bugs

■ Using Debuggers
  + Familiar or fancy, they’re still WYSIWYG
    – Human intensive hunting
    – But where did the bug come from!?

■ Incremental Parallel Programming
  + Take things one step at a time
    – Human intensive
    – How patient are you? Some steps are big!

Using dbx for Guided programs

atlas % setenv OMP_NUM_THREADS 2
atlas % dbx a.out
dbx version 3.11.8
test: 25 CALL mppbeg
(dbx) stop in __release_join_bar
[2] stop in __release_join_bar
(dbx) run
[2] thread 0xffffffff81af52c0
stopped at
[__release_join_bar:721,
  0x12001a198]
Using dbx for Guided programs

- Inquire what threads are doing with tstack
- Switch to other thread with tset
- Continue running

Better Tactic For Bugs

- **Assure** systematically finds Communication Leaks
  - Identifies source of bug as well
  - Finds non-deterministic errors
  - Trades computer time for human time
**What is Assure**

- **Assure** for validating **OpenMP**
  - Analyze
  - Restructure
  - Test
  - Improve
  - Q/A

**Secondary Use**

**Primary Use**

**Input:** Program with OpenMP directives; test data.

**Output:** Report of communication leaks.

---

**Why isn’t this automated?**

(Ans: It is. E.g., **Assure**.)

- Guide provides compile time diagnostics
  - Limits: subroutine being compiled and heuristic

- What is Assure?
  - Validates parallel programs
  - Missing or incorrect scoping is invalid

- What does Assure do?
  - Simulates parallel run
  - Finds communication leaks!
What Is Parallel Validation?

- **Assure** identifies *incorrect behavior for parallel program*
- Define correct behavior? **Assure** uses --
  - Parallel program and a provided data-set
- When validated, a program is valid --
  - For any number of processors at runtime
  - For all execution timing variations
  - Across supported platforms

Verifying Storage Class Choice

- Misclassified x and y as shared
- **Assure** report:
  Storage conflicts in PARALLEL DO, DSQ/2 (dsq.f):

```plaintext
Conflict   Source   Source - Type Symbol Sink
----------------- --------------------------
Read -> Write X   DSQ/7 - 5
Read -> Write Y   DSQ/7 - 6
Write -> Write X   DSQ/5
Write -> Write Y   DSQ/6
```

Code snippet:

```plaintext
01: subroutine dsq(a, b, c, n)
02: c$omp parallel do private(i)
03: c$omp& shared(a, b, c, n, x, y)
04: do i = 1,n
05:   x = a(i) - b(i)
06:   y = b(i) + a(i)
07:   c(i) = x * y
08: end do
```
Verifying Storage Class Choice

- Misclassified a and b as private

```fortran
01: subroutine dsq(a, b, c, n)
02: c$omp parallel do shared(c, n)
03: c$omp& private(i, a, b, x, y)
04:  do i = 1, n
05:   x = a(i) - b(i)
06:   y = b(i) + a(i)
07:   c(i) = x * y
08:  end do
```

Assure report:

Errors in PARALLEL DO, DSQ/2 (dsq.f):

- Error: DSQ/5 (dsq.f): uninitialized read of PRIVATE symbol 'B' in PARALLEL DO
- Error: DSQ/5 (dsq.f): uninitialized read of PRIVATE symbol 'A' in PARALLEL DO

Assure Example

- Invalid data bug:

```fortran
    do 10 n=1,1000
 10    A(n) = 0.0
    c$omp parallel do private(B,i,j)
    c$omp& shared(A,m,a1,a2,b1,b2)
    do i=1,m
       do 20 j=a1(i),a2(i)
 20      A(j) = (expression)
       do 30 j=b1(i),b2(i)
 30      B(j) = (Reference to A(j))
    end do
    print *,B
```

Valid Data --

<table>
<thead>
<tr>
<th>m=4</th>
<th>i</th>
<th>a1</th>
<th>a2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

Invalid Data --

<table>
<thead>
<tr>
<th>m=4</th>
<th>i</th>
<th>a1</th>
<th>a2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>
Assure Example

- Invalid data bug:

```
01:   do 10 i=1,20
02: 10   A(i) = 0.0
03:  c$omp parallel do private(B,i,j)
04:  c$omp& shared(A,a1,a2,b1,b2,m)
05:   do i=1,m
06:   do 20 j=a1(i),a2(i)
07: 20   A(j) = (Expression)
08:   do 30 j=b1(i),b2(i)
09: 30   B(j) = (Func of A(j))
10:  end do
11:   do 40 i=1,20
12: 40   print *, B(i)
```

- Assure Report:

Errors in ROUTINE, ITER/1 (iter.f):
-----------------------------------------------
Error:  ITER/9 (iter.f):  PRIVATE
symbol 'B' referenced outside parallel construct as 'B' in ITER/12
Storage conflicts in PARALLEL DO,
ITER/16 (iter.f):
Conflict Source Source - Sink
Type Symbol Routine/Line
Write -> Read A ITER/7 - 9

AssureView: Call Graph
AssureView: Common Mismatch

Kuck & Associates, Inc.

AssureView: Errors

Kuck & Associates, Inc.
GuideView visualizes OpenMP

- Analyze
- Restructure
- Test
- Improve
- Q/A

What is GuideView

Input: Program with OpenMP directives; test data.
Function: Navigate complex problem down to performance problems
Example Profile

Each sample covers 4.00 byte(s) for 8.3e-06% of 11743.0342 seconds

<table>
<thead>
<tr>
<th>%time</th>
<th>seconds</th>
<th>cum</th>
<th>% cum sec</th>
<th>procedure (file)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>1003.0098</td>
<td>8.5</td>
<td>1003.01</td>
<td>proj_slv_node_on_mr_seg_</td>
</tr>
<tr>
<td>4.7</td>
<td>548.3232</td>
<td>17.9</td>
<td>2102.68</td>
<td>shell_force_el_pl_4p_</td>
</tr>
<tr>
<td>4.4</td>
<td>522.3145</td>
<td>22.4</td>
<td>2624.99</td>
<td>pkshell_egy_el_pl_v__</td>
</tr>
<tr>
<td>3.8</td>
<td>450.5781</td>
<td>26.2</td>
<td>3075.57</td>
<td>shell_lam_stress_pl_3d_</td>
</tr>
<tr>
<td>3.3</td>
<td>382.9053</td>
<td>40.2</td>
<td>4718.43</td>
<td>pkcalc_node_vel__</td>
</tr>
<tr>
<td>3.2</td>
<td>372.4639</td>
<td>43.4</td>
<td>5090.90</td>
<td>pkcalc_node_accel__</td>
</tr>
<tr>
<td>3.1</td>
<td>365.8145</td>
<td>46.5</td>
<td>5456.71</td>
<td>shell_stress_el_pl_</td>
</tr>
<tr>
<td>2.6</td>
<td>306.9453</td>
<td>60.8</td>
<td>7144.42</td>
<td>pkcontact_node_trial_vel_posn_1_</td>
</tr>
<tr>
<td>2.4</td>
<td>276.1680</td>
<td>63.2</td>
<td>7420.58</td>
<td>trans_lvecs_to_gvecs__</td>
</tr>
<tr>
<td>2.3</td>
<td>270.2891</td>
<td>65.5</td>
<td>7690.87</td>
<td>spin_wait_join_barrier</td>
</tr>
</tbody>
</table>

GuideView adds focus

- Visualize the performance report
- Pie Chart --
  - Percent parallel
  - Overhead
    - Locks
    - Barriers
    - Imbalance
- Bar Chart --
  - Amdahl’s Law
**GuideView** adds focus

- Parallel regions
- Sorted by --
  - Overhead
  - Time
- Compare --
  - Parallel time
  - Synched time
  - Lock time
  - Barrier time

---

**GuideView** adds focus

- Per Region
  - & Per Thread
- High Barrier Time!
- Load Imbalance
  - Possible

(Note: this is a small run)
# DGauss Performance

<table>
<thead>
<tr>
<th>Job</th>
<th>Intel Pentium Pro (200Mhz)</th>
<th>Cray J90</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$P=1$</td>
<td>$P=4$</td>
</tr>
<tr>
<td>Job 1</td>
<td>54</td>
<td>20</td>
</tr>
<tr>
<td>Job 2</td>
<td>491</td>
<td>194</td>
</tr>
<tr>
<td>Job 3</td>
<td>3241</td>
<td>1256</td>
</tr>
<tr>
<td>Job 4</td>
<td>13362</td>
<td>5700</td>
</tr>
</tbody>
</table>

# MM5 Performance

<table>
<thead>
<tr>
<th></th>
<th>Small Problem</th>
<th>Large Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Digital Alpha (400Mhz)</td>
<td>Digital Alpha (400Mhz)</td>
</tr>
<tr>
<td></td>
<td>Unix</td>
<td>NT</td>
</tr>
<tr>
<td>Serial</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$P=1$</td>
<td>1.0 (760)</td>
<td>1.0 (701)</td>
</tr>
<tr>
<td>$P=2$</td>
<td>1.87</td>
<td>1.82</td>
</tr>
<tr>
<td>$P=3$</td>
<td>2.59</td>
<td>2.46</td>
</tr>
<tr>
<td>$P=4$</td>
<td>3.25 (234)</td>
<td>3.03 (233)</td>
</tr>
</tbody>
</table>
Parallel Software Engineering needs to be addressed to produce quality applications.

OpenMP solves the portability problem.

The KAP/Pro Toolset for OpenMP is available now!

http://www.openmp.org