



Intel Cluster OpenMP*

Gelato ICE San Jose 2007

April 17, 2007

Larry Meadows, Intel SSG/DPD/PAT

* Other names and brands may be claimed as the property of others.



What is OpenMP*?

- OpenMP is a language for annotating sequential C, C++ and Fortran programs. Annotations produce:
 - Creation of thread teams
 - Execution in parallel
 - Redundant execution
 - Cooperative execution
 - Synchronization between threads
 - Barriers, locks, critical sections, single threaded regions
 - Creation of variables private to a thread
- Implemented through compiler support and a run-time library

* Other names and brands may be claimed as the property of others.



Cluster OpenMP*

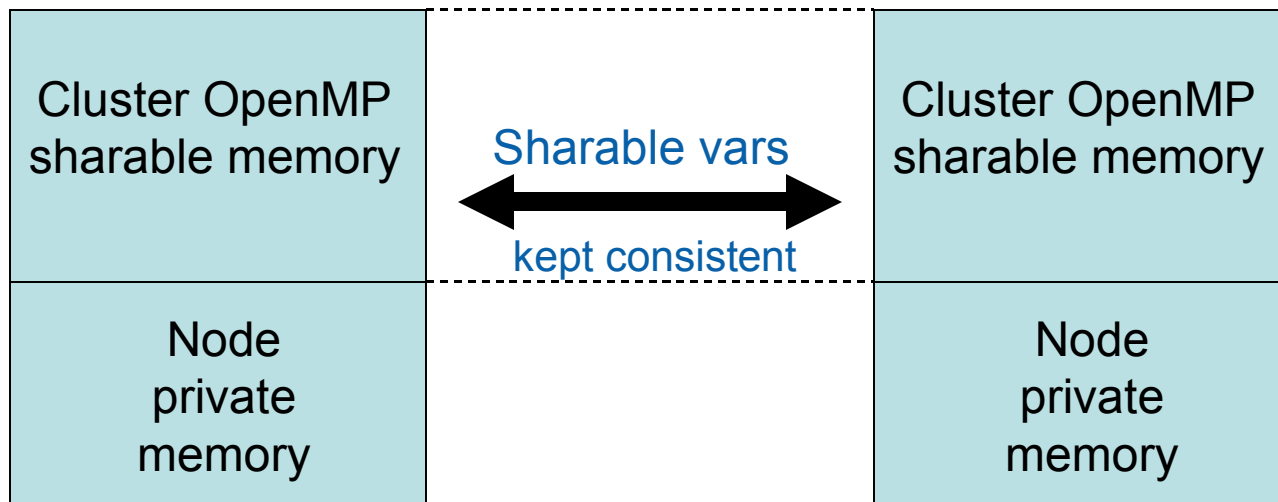
- Cluster OpenMP is a run-time library that supports running an OpenMP program on a cluster.
- It is released with the Intel compilers (starting with 9.1) and requires an extra license.
- Suitable Programs:
 - Programs that scale successfully with OpenMP on SMP
 - Programs that have good data locality
 - Programs that use synchronization sparingly

Why should you use Cluster OpenMP*?

- To run a program on many processors:
 - Old options:
 - Buy a large shared memory machine and use OpenMP
 - Hardware: expensive; Programming: inexpensive
 - Buy a cluster and use MPI
 - Hardware: inexpensive; Programming: expensive
 - New option:
 - Buy a cluster and use OpenMP
 - Hardware & Programming: inexpensive

Cluster OpenMP* Memory Model

If multiple OpenMP threads access a variable, it **must** be sharable!



Process 0 address
space

Process 1 address
space

Cluster OpenMP* Sharability

- All variables in a **shared** clause or implicitly shared must be made sharable, except for system variables like file pointers¹, etc.
- Sharability may be determined automatically by the compiler, or with a compiler option.
- Sharability is a Cluster OpenMP concept; it can be specified by the user with a directive:

```
#pragma intel omp sharable(var) // C,C++  
!dir$ omp sharable(var) ! Fortran
```

¹Cluster OpenMP does not provide a single system image.

Porting: Making the shared variables sharable

- Handled automatically
 - Stack variables in a routine where they are shared
- Not handled
 - Stack variables in a different routine from where they are shared
 - Heap variables shared anywhere
 - Static variables shared anywhere
- Tools for finding the variables not handled:
 - Compiler interprocedural analysis traces shared use of arguments back through call-sites to allocation sites
 - Runtime check catches the shared use of node-private heap variables
 - Compiler options for Fortran COMMONs, etc
 - #define for C
 - #define malloc kmp_sharable_malloc
 - #define free kmp_sharable_free
 - Etc.

A Simple Cluster OpenMP* Program

```
#include <omp.h>
static int x;
#pragma intel omp sharable(x)

int main()
{
    x = 0;
#pragma omp parallel
    {
#pragma omp critical
        x++;
    }
    printf("%d should equal %d\n", omp_get_max_threads(), x);
}
```

* Other names and brands may be claimed as the property of others.



Compiling and Running the Program

```
$ icc -cluster-openmp test.c
```

Assume there are two nodes, **rufus** and **dufus**, and we wish to run one process per node with 4 threads each.

```
$ cat kmp_cluster.ini  
--hostlist=rufus,dufus --processes=2 --process_threads=4  
MY_ENV_VAR=value
```

For more info on init file, see User's Guide.

The Cluster OpenMP* runtime reads **kmp_cluster.ini** and determines the parameters: host names, number of processes, and number of threads per process.

```
$ a.out  
8 should equal 8
```

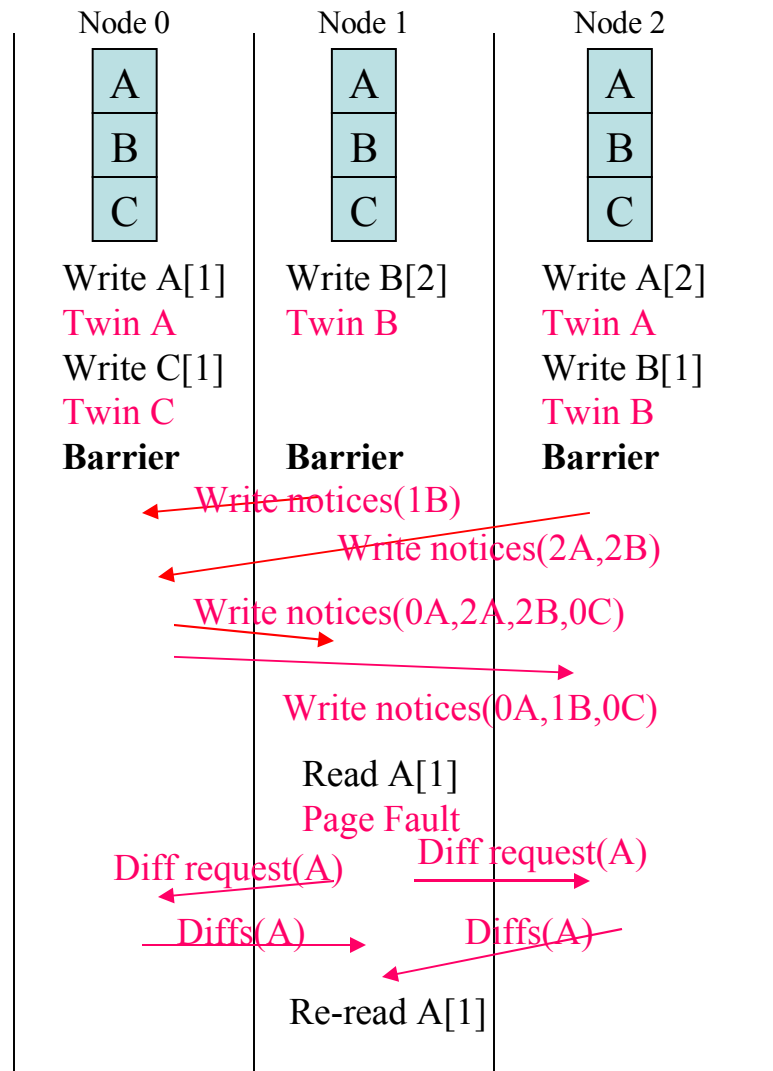
* Other names and brands may be claimed as the property of others.

Consistency Protocol

- Basic idea:
 - When a page of sharable memory is not up-to-date, it becomes *protected*
 - An access faults (SIGSEGV) into our software which requests info from remote nodes to update the page
 - Protection is removed from page
 - Instruction causing the fault is re-started, this time successfully accessing the data

Consistency Protocol Detail

Write notice notation -
1B means node 1
wrote page B.



Tools for Use with Cluster OpenMP

- **Correctness Tools:**

- Intel® Thread Checker
 - Finds variables that should be made sharable
- Intel® Compiler
 - -clomp-sharable-propagation option finds variables that should be made sharable
 - Fortran options make all variables with certain attributes sharable
- Disjoint heap
 - Find allocated variables that should be sharable
- IDB
 - Debugger now supporting printing the value of sharable variables

- **Performance Tools:**

- Segvprof
 - Shows where SEGV hot spots are when executing the code
- Intel® Trace Analyzer/Collector
 - Displays message traffic between nodes
- Cluster OpenMP dashboard
 - Real-time view of page state and message volume
- Intel® Thread Profiler
 - Displays OpenMP-related performance information

Correctness Tools:

Intel® Thread Checker

- Support in version 3.1 as a preview feature.
- Used as follows:
 - `setenv TC_PREVIEW 1`
 - `setenv TC_OPTIONS [verbose,]shared`
 - `setenv KMP_FOR_TCHECK 1`
 - Setup `kmp_cluster.ini` with "`--processes=1 --process_threads=4`"
 - As usual, setup proper `LD_LIBRARY_PATH` to use latest `libclusterguide.so` (from 10.0 compiler)
 - Compile the application with `-g`
 - `tcheck_cl [-c] <executable> <executable args>`
 - `-c` option clears the instrumentation cache, forcing re-instrumentation of all libraries

Intel® Thread Checker output

ID	Short Description	Severity	Context	Description	1st Access	2nd Access
1	Data Shared	Error	1	"c_ex2.c":21 "c_ex2.c":21 and a different thread at Unknown is not sharable	Unknown	"c_ex2.c":21
2	Data Shared	Error	1	"c_ex2.c":21 "c_ex2.c":21 and a different thread at "c_ex2.c":21 is not sharable	"c_ex2.c":21	"c_ex2.c":21
3	Data Shared	Error	1	"c_ex2.c":85 "c_ex2.c":88 and a different thread at Unknown is not sharable	Unknown	"c_ex2.c":88
4	Data Shared	Error	1	"c_ex2.c":10 "c_ex2.c":101 and a different thread at "c_ex2.c":46 is not sharable	"c_ex2.c":46	"c_ex2.c":101

Correctness Tools:

Compiler option: `-clomp-sharable-propagation`

- Compile all sources

- `ifort -cluster-openmp -clomp-sharable-propagation -ipo file.f file2.f`

- At the “link” step, sharable directive warnings will indicate variables that must be made sharable

- `fortcom: Warning: Sharable directive should be inserted by user as`

- `'!dir$ omp sharable(n)' in file file.f, line 23, column 16`

Correctness Tools: Fortran compiler options

Original code

use this option

makes it as if you wrote this code

```
common /blk/ a(100)
```

```
common /blk/ a(100)  
!dir$ omp sharable (/blk/)
```

`-clomp-sharable-commons`

```
real a(100)  
save a
```

```
real a(100)  
save a  
!dir$ omp sharable (a)
```

`-clomp-sharable-localsaves`

```
module m  
real a(100)
```

```
module m  
real a(100)  
!dir$ omp sharable (a)
```

`-clomp-sharable-modvars`

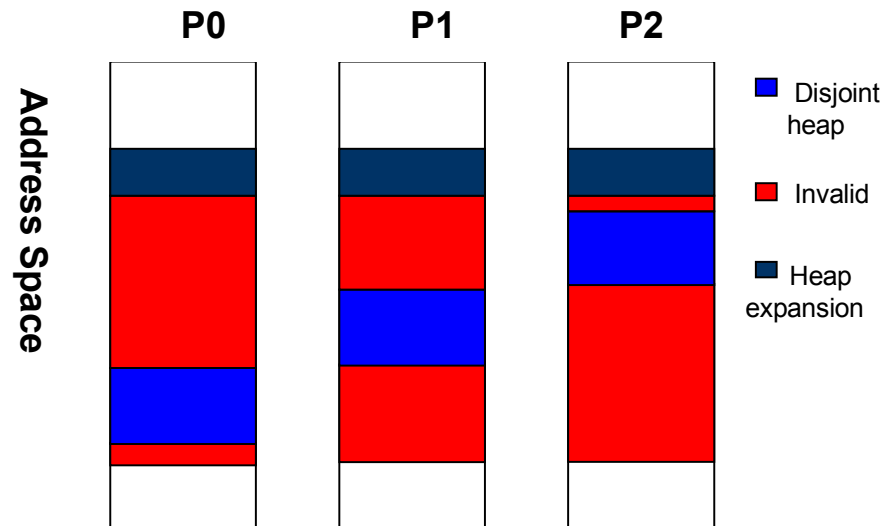
Correctness Tools:

Runtime Disjoint heap

- Compile with “-g”
- `setenv KMP_DISJOINT_HEAPSIZE <size>`
- Run program
- If you get an error:
 - Cluster OMP Fatal: Proc#0 Thread#2 (INITIAL): Segmentation fault (ip=0x400c2f address=0x5a85c20)
 - Cluster OMP Fatal: Proc#0 Thread#2 (INITIAL): This is an address in the local heap for process 1

- Use `addr2line` to find source line

```
% addr2line -e switch.exe 0x400c2f  
/home/jhcownie/tmp/switch.c:17
```



Porting Process

- Make sure the program works with OpenMP*
- Try the program with Cluster OpenMP
 - If it works – done
 - If not, then...

all
langs

- **Use Intel® Thread Checker**
- Use `-clomp-sharable-propagation`
- Use runtime disjoint heap
- Fortran: try Fortran options

stack / static alloc
heap alloc

static
alloc

* Other names and brands may be claimed as the property of others.

Correctness Tools:

IDB

- `setenv IDB_PARALLEL_SHELL <path to your ssh>`
 - E.g. `"setenv IDB_PARALLEL_SHELL /usr/bin/ssh"`
- Source the `idbvars.csh` script in `/opt/intel/idb/<platform>/idbvars.csh`
- Add the following to `kmp_cluster.ini`:
 - `"--no_heartbeat --IO=system --startup_timeout=60000"`
- Compile the application with `-g`
- Setup `IDB_HOME` to be the directory where `idb` resides
 - E.g. `"setenv IDB_HOME /opt/intel/idb/<platform>/"`
- Use `-clomp` and full pathname to executable
 - E.g. `"idb -clomp ~/Scrap/simp.exe"`

IDB Tips for Cluster OpenMP

- The program begins running automatically when you enter IDB.
- When it prints a prompt, that means it is at an automatic breakpoint set in the runtime library. So, you don't "run" the application – instead you "continue" from that point.
 - IDB team may be changing the product so that the first prompt inside IDB looks like a breakpoint, to give the user a hint of what is happening.
- "print" is available for printing the value of sharable variables
- "break" is also available
- The functions for showing OpenMP team information are not yet supported for Cluster OpenMP.

Performance Tools:

segvprof.pl

- Compile with `-g` to get line level profiles
- Set the environment variable `KMP_CLUSTER_PROFILE` to 1
- Run the code
- Analyze the `*.gmon` files with `segvprof.pl`
% `segvprof.pl -e <executable> *.gmon`
- Report shows the number of SEGVs which occurred at each line
- Sorted by region if `-cluster-openmp-profile` was used at compile time otherwise for the whole program

Sample use of SEGV profiling

Write SEGVs:

Count	Proc	Source Location/Library	Line 109:
1024	1	bp_para3d-23.c:109	imgfun (n+naux, jvox, ivox) +=
1024	3	bp_para3d-23.c:109	U*U * (
1024	2	bp_para3d-23.c:109	(1.0f-p-q+pq) *projfun(k, j +i)+

Fetch SEGVs:

Count	Proc	Source Location/Library
6798	2	bp_para3d-23.c:109
6798	3	bp_para3d-23.c:109
6798	1	bp_para3d-23.c:109
3762	1	bp_para3d-23.c:109
3762	3	bp_para3d-23.c:109
3762	2	bp_para3d-23.c:109
1024	3	bp_para3d-23.c:109
1024	2	bp_para3d-23.c:109
1024	1	bp_para3d-23.c:109

```
U*U * (  
(1.0f-p-q+pq) *projfun(k, j +i )+  
(q-pq) *projfun(k, j +i1)+  
(p-pq) *projfun(k, j1+i )+  
(pq) *projfun(k, j1+i1) );
```

RED is written and read

GREEN is read

Note that 0 has no SEGV

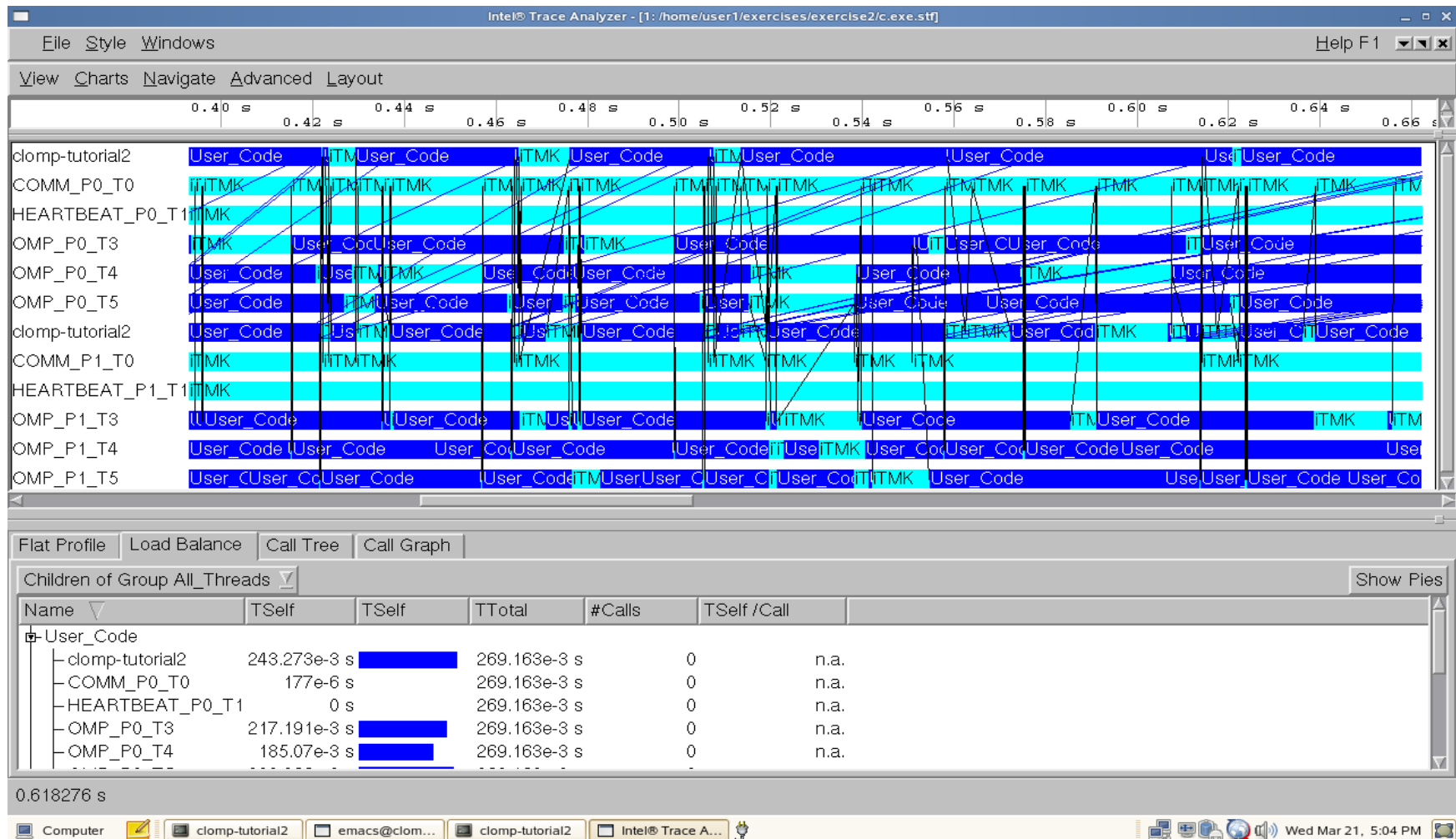
Performance Tools:

Intel® Trace Analyzer / Collector

- `setenv KMP_TRACE 1`
- Run application
- `traceanalyzer <application>.stf`

- Add user events to your code:
 - `VT_funcdef(char * name, 0, int * handle)`
 - Ex: `VT_funcdef("INIT", 0, &init_handle);`
 - `VT_begin(int handle);`
 - Ex: `VT_begin(init_handle);`
 - `VT_end(int handle);`
 - Ex: `VT_end(init_handle);`
- Show the event timeline and then ungroup the Application thread to reveal the user-defined functions on the timeline and see the times listed in the thread window.

Output of Intel® Trace Analyzer



Performance Tools:

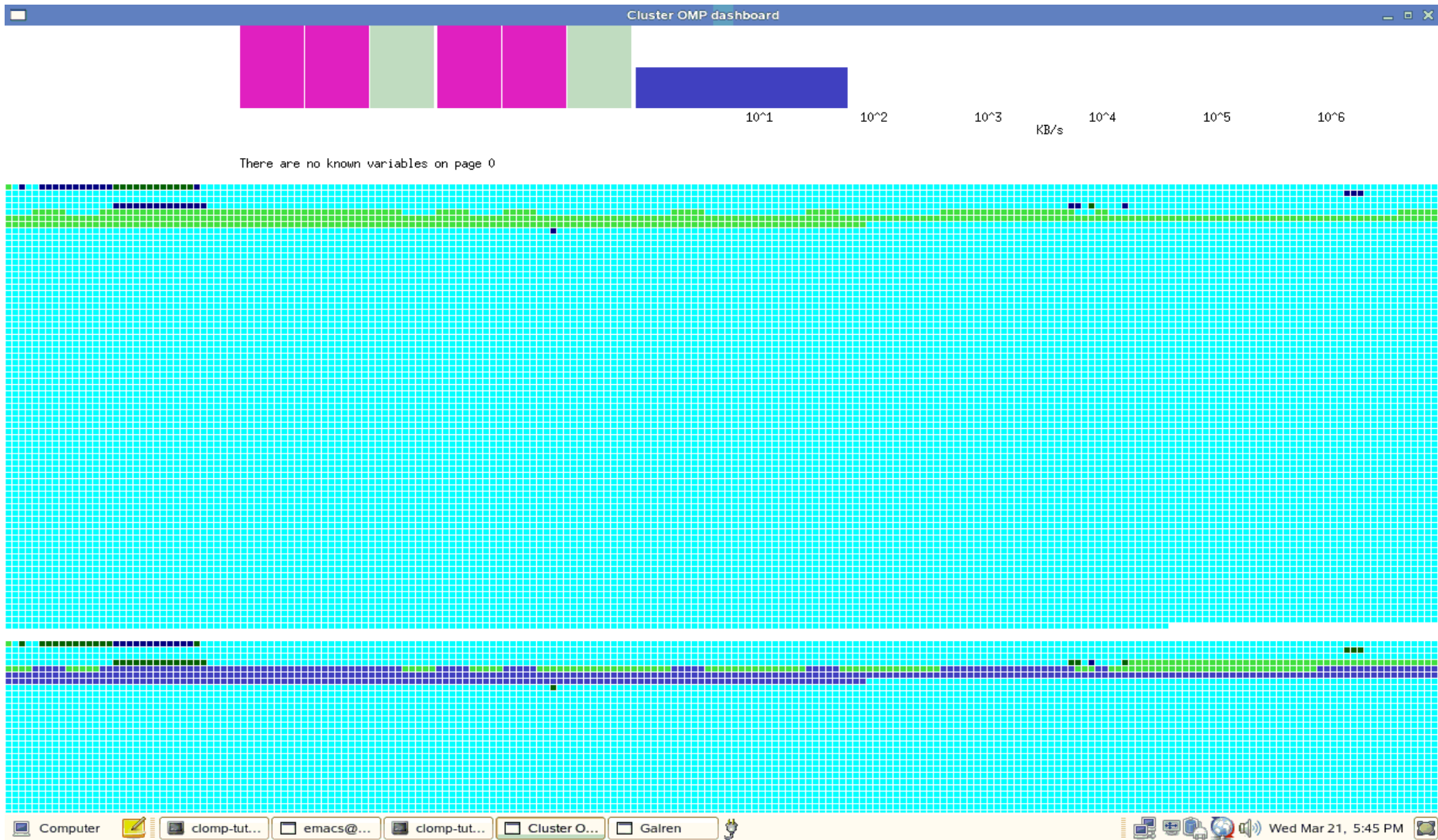
Cluster OpenMP Dashboard

- `setenv KMP_CLUSTER_DISPLAY :0.0`
- Add the following to the `kmp_cluster.ini` file:
 - `KMP_CLUSTER_DISPLAY=:0.0`
 - For example:

```
--processes=2 -process_threads=4 -launch=ssh  
KMP_CLUSTER_DISPLAY=:0.0
```

- Run the application as normal

Output of the Cluster OpenMP Dashboard



Cluster OpenMP Dashboard

Keyboard commands:

r - page fault (read) mode
w - page fault (write) mode
b - page fault (r/w) mode
t - toggle page fault/page state mode
z - zoom out display
j - scroll display down
k - scroll display up

Mouse commands:

Left click/hold and drag to zoom
Left click and release on a page
to see trace information and
registered variables

Performance Tools:

Intel® Thread Profiler

- Compile with `-cluster-openmp-profile`
- Creates a file `guide.gvs`
- Open that file using the Windows* GUI
- There is a lot more information in `guide.gvs` and the file is readable, so advanced users can take a look.
- Use `-p` with `clomp_forecaster.pl` to create a sorted, summarized CSV file from the `.gvs` file, for use with a spreadsheet.



"Intel", and "Thread Profiler" are registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

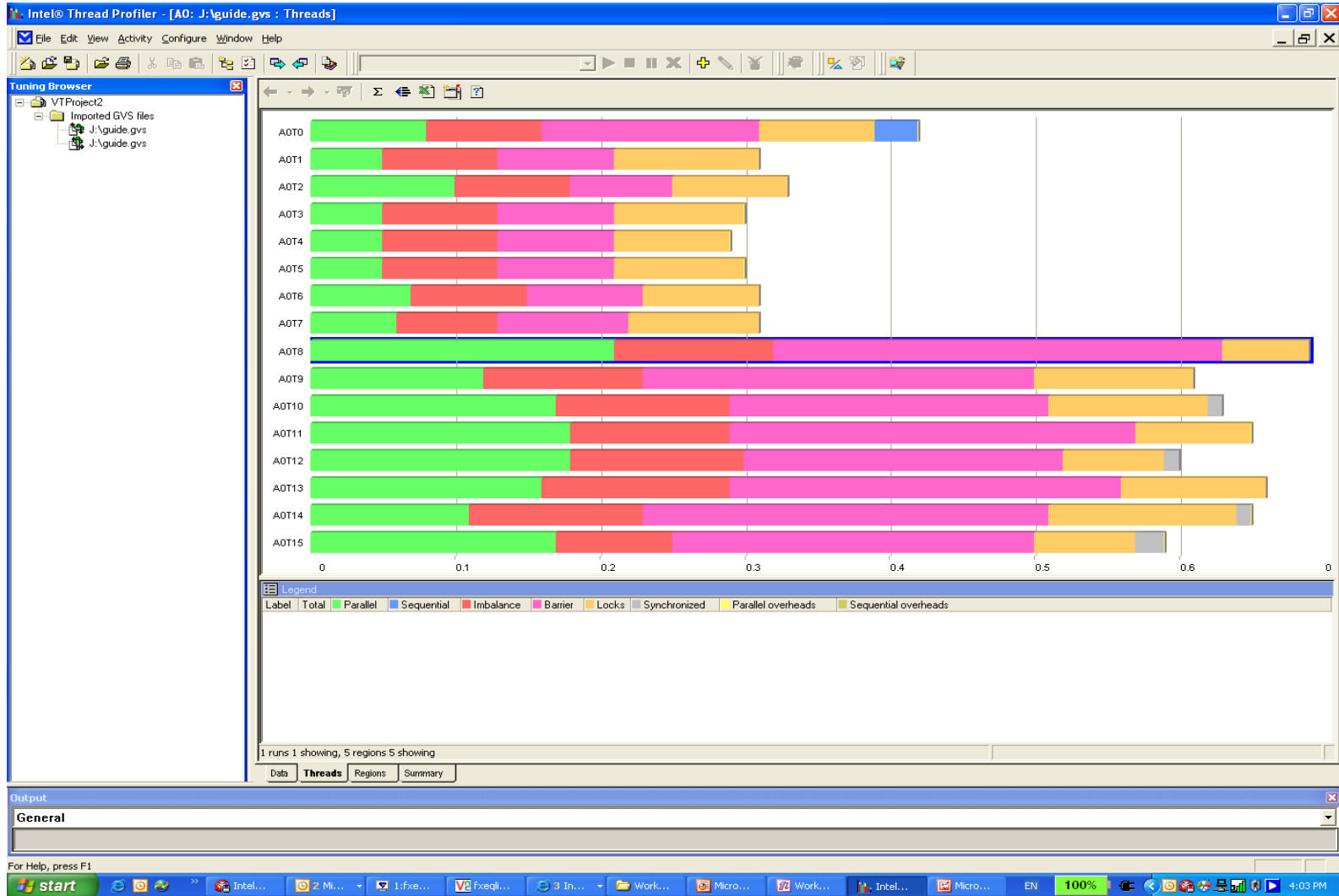
Software and Solutions Group

28

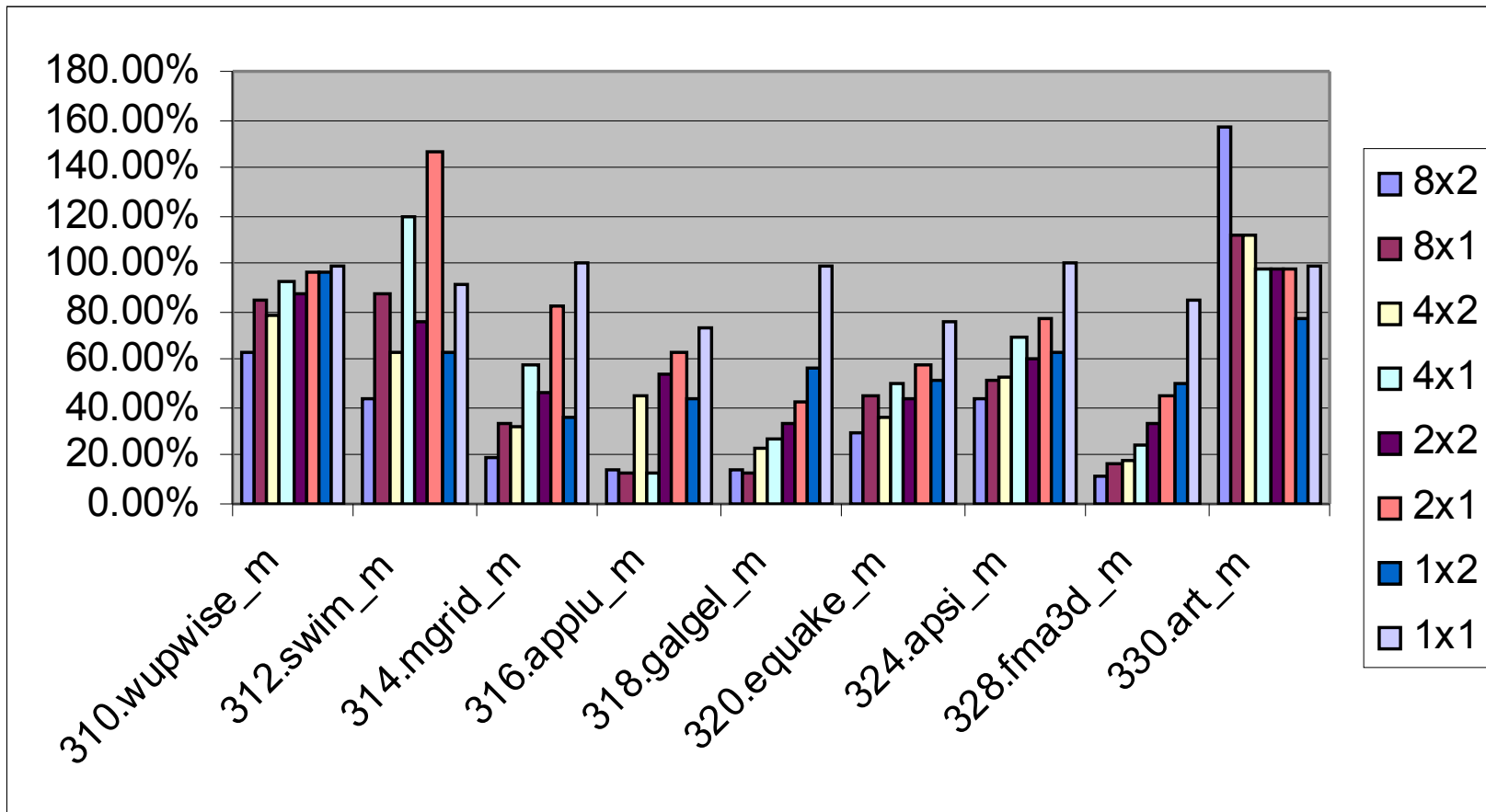


* Other names and brands may be claimed as the property of others.

Intel® Thread Profiler output



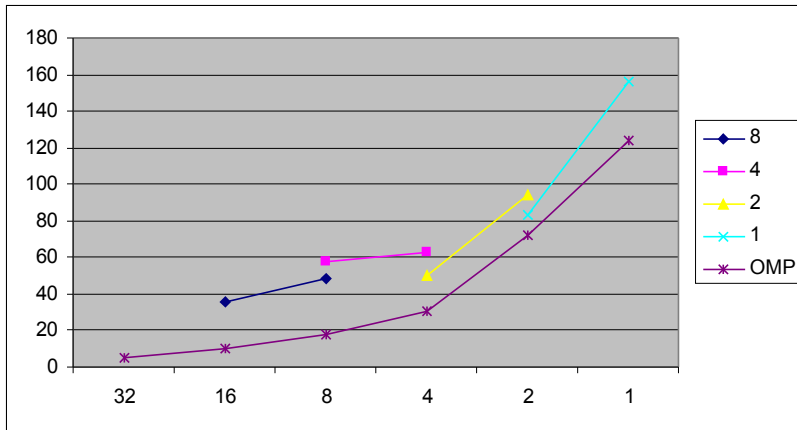
SPECOMPM* performance vs. SMP



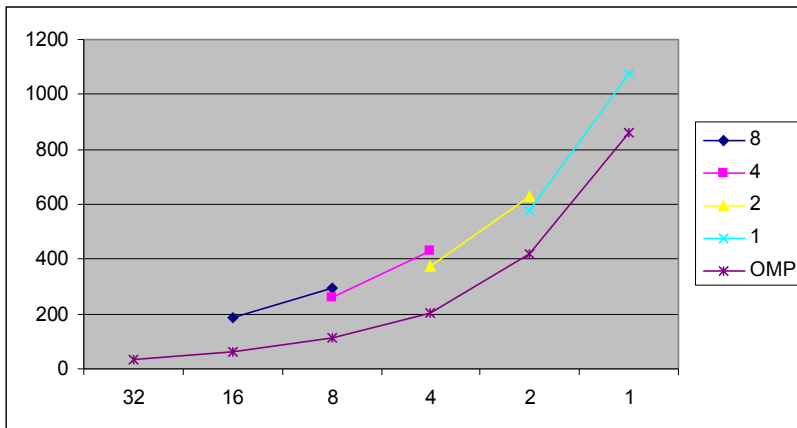
Ratio of performance for same number of threads
> 100% is better (Cluster OpenMP was faster)
< 100% is worse (Cluster OpenMP was slower)

NASA flowcart CFD code: Performance vs. SMP

1,2,4, or 8 nodes
1 or 2 threads/node
Times in seconds for 10 iterations



Small wing simulation



Large wing simulation



Dynamic Sharable Allocation

- C API – just like malloc/free

```
#include <omp.h>
void *ptr;
ptr = kmp_sharable_malloc(size);
ptr = kmp_sharable_realloc(ptr, size);
ptr = kmp_sharable_calloc(n, size);
kmp_sharable_free(ptr);
```

- Fortran API

```
include 'omp_lib.h' ! or use omp_lib
      real, allocatable :: a(:)
!dir$ omp sharable(a)
      allocate(a(count))
Cray pointer case
      real a(100); pointer (p,a); p = kmp_sharable_malloc(400)
```

C++ Dynamic Sharable Allocation

- In all cases, use `#include <kmp_sharable.h>`
- To make all objects of certain class sharable
 - Convert `class foo { ... };` to
`class foo: public kmp_sharable_base { ... };`
- To make instances of objects sharable
 - Convert `bah *p=new bah (...);` to
`bah *p= new kmp_sharable bah(...);`
- To make STL containers and contents sharable
 - Convert `vector<int> *vp=new vector<int>;` to
`vector< int,kmp_sharable_allocator<int> > *vp= new
kmp_sharable vector<int,kmp_sharable_allocator<int> >;`

Cluster OpenMP* Resources

- Cluster OpenMP User's Guide
 - <compiler root>/doc/cluster_omp_docs/UsersGuide.pdf
- Example programs and tools
 - <http://premier.intel.com>, [clomp_tools.tar.gz](#) under:
 - Intel C++ Compiler, Linux* Cluster OpenMP*
 - Intel Fortran Compiler, Linux* Cluster OpenMP*
 - Contains
 - `clomp_getlatency.pl` – [measure latency to remote node](#)
 - `clomp_configchecker.pl` – [check configuration](#)
 - `clomp_forecaster.pl` – [estimate program performance](#)
 - `segvprof.pl` – [display per line/function profile information](#)
 - Example codes and `kmp_cluster.ini` file

* Other names and brands may be claimed as the property of others.



Ignoring SEGVs in Various Debuggers

- idb: it's automatic
- gdb: in `.gdbinit` file:
 - `handle SIGSEGV nostop noprint`
- Etnus TotalView®: in `.tvdrc` file:
 - `dset TV::signal_handling_mode {Resend=SIGSEGV}`