Introduction: OpenMP is clear and easy ..

😊 Easy directive based handling

😊 Portable code

😊 Single source for sequential/parallel

😊 Incremental parallelization

😊 Well-defined semantics

So, why tools?
Introduction: ... but

- Performance tuning can be tricky
  Losses are transparent

- `Easy´ to introduce data conflict bugs
  These are tedious to fix

- Data mapping
  Will be important for clusters. Heavy inter-thread data
  dependencies often difficult to detect.

- OpenMP tools must
  - bring out the strengths of OpenMP
  - help to overcome the problems

Introduction: KAP/Pro Toolset

- Supported Platforms:
  - Compaq alpha (True64 Unix, WinNT4.0)
  - SGI MIPS (IRIX 6.5)
  - Sun SPARC (Solaris 2.5, 2.6, 2.7)
  - Intel IA32 (WinNT4.0, Linux86, Solaris86)
  - HP PA-RISC 2.0 (HP-UX 11)
  - IBM RS/6000 (AIX 4.1.* - 4.3)
Overview: KAP/Pro Toolset

- Guide: fully OpenMP compliant F77/F90/C/C++ compiler
- GuideView: graphical performance analysis tool
- Assure: fully OpenMP compliant F77/F90/C/C++ compiler for checking semantical code correctness
- AssureView: graphical backend of Assure
- PerView: online performance monitor

Overview: Guide and GuideView

- Guide: fully OpenMP compliant C/C++/F77/F90 compiler
- Based on native compiler => native performance, native options are passed to backend compiler

PLUS:

- An additional statistics library on user request, traces OpenMP events at runtime
- After run, visualize tracefile with GuideView
Overview: Sample with GuideView

What can be detected?

Guide (and its alarm colors):

- Load imbalance
- Too many synchronization points
- Too much ‘communication’
- Heavy critical sections
- Acquiring locks
- Heavy sequential sections (overhead)
- Idle times

And the good color is
(independent work of thread)
Performance tuning: invoke Guide -WGstats

- Compile code with Guide compiler:
  
  `guidef90 [options] -WGstats -o myprog myprof.f90`

- Execute
  
  ```
  setenv OMP_NUM_THREADS ..
  ./myprog
  => Guide statistics file `guide.gvs` has been produced
  ```

- Visualize
  
  `guideview guide.gvs` &

What can be detected: Loadimbalance

```
!$OMP PARALLEL
!$OMP DO
DO i=1,xlen
   IF( i<xlen/2 ) THEN
      x(i)=sin(float(i))
y(i)=real(i)
   ELSE
      x(i)=1.
y(i)=1.
   ENDIF
ENDDO
!$OMP END PARALLEL
```
© Pallas GmbH

Source code view is available!

Right mouse click => Show code

What can be detected: Barriers

```fortran
!$OMP PARALLEL
!$OMP DO
DO i=1,xlen
    z(i) = x(i-1)+y(i)
ENDDO
!$OMP END PARALLEL
```

```
DO i=2,xlen
    z(i) = x(i-1)+y(i)
ENDDO
```
What can be detected: Overhead

Parallelization of innermost loop:

DO i=1,1000

!$OMP PARALLEL DO
DO j=1,100
  temp(j) = & 
  temp(j)+1./i
ENDDO

ENDDO

What can be detected: Critical + Locks

 !$OMP PARALLEL PRIVATE(temp)
 !$OMP DO
DO i=1,100000
  ! PARALLEL PRIVATE
  ! CALCULATIONS ..
  temp = 1.
  ! CRITICAL UPDATES
  !$OMP CRITICAL
  sum = sum+sin(temp)
 !$OMP END CRITICAL
ENDDO
 !$OMP END PARALLEL
The incremental optimization cycle

- Initial OpenMP program
- `guidef -WGstats`
- `guideview`
- Tune most important region(s)
- O.K.
- Assure (View)
- fault

Guide

Output file names can be controlled by environment variables:

- `KMP_STATSFILE` (guide.gvs file)
- `KDD_OUTPUT` (.kdd file of assure, see below)

Three meta characters are available to use in the names:

- `%P` #threads of the run
- `%I` unix process id of the run
- `%H` hostname to run the program
Guide

E.g.

setenv KMP_STATSFILE gs%P_%I_%H

gives Guide statsfile similar to

    gs2_11074_vision.gvs

GuideView: Displays

Usage:

guideview <stats_file>

Main display

- Estimated speedup

- Whole program timing breakdown
GuideView: Displays

Whole timing breakdown (averages of threads)

Left button doubleclick => ...

GuideView: Displays

... => Thread timing breakdown

Region View => ...
GuideView: Displays

...=> Region timing breakdown (again average or per thread)

Each parallel region (denoted `R1, R2, ...´)

Each sequential part (denoted `S1, S2, ...´)

Inside each region (optional):

Breakdown into sections between barrier points (denoted `RxB1, RxB2 ...´).
Can be unselected in "Options/Show Barrier Regions"
GuideView: Displays

Always remember:

There is a link between each of the displayed sections and the corresponding code

(right mouse button on column => context `show code`)

GuideView: Displays

- Click "Help/GuideView Info" for explanations
GuideView: Displays

**Sort button**
Several sort criteria for the regions, in particular:
- sort by time => most important first
- sort by overhead => most critical first

**Options/Filter button**
Filter regions, e.g. only display heavy ones (more than 10% of total)

GuideView: Trace Comparison

Compare different #threads for same program (e.g.)

- `guideview <first statsfile>`

- Click "File/open new file"
  - Menu opened => select second statsfile

All views are then comparative between the two files
GuideView: Trace Comparison (1 <-> 2 threads)

- Estimated Speedup
  - upper speedup curve
  - lower speedup curve
  - relative speedup curve
  - no speedup
  - no initial speedup

Whole Program Time Distributions

Overview: Assure and AssureView

- **Assure77/f90**: OpenMP compliant, restriction in usage of OMP library (in particular: OMP_GET_THREAD_NUM)
- Use as normal compiler, but not for getting performance (small input data set)
- Multithreaded run is simulated sequentially, all memory accesses verified
- Run AssureView to visualize error breakdown. When "No Errors" are reported, multithreaded run is assured free of semantical errors as explained below, but only in the branches touched by the simulation run.
What can be detected: invoke Assure

- Compile code with yet another compiler:
  `assuref90 [options] -o myprog myprof.f90`

- Execute (but don't expect performance!!)
  `./myprog`

- Visualize
  `assureview`

Assure

The Assure process

- Source program
- Assuref compiler
- Run executable
- Simulation file .kdd
- Project file .prj
- Assureview
What can be detected: Conflicts

real:: a(0:N), b(N)
a(0) = 0.
!$OMP PARALLEL
!$OMP DO
DO i=1,N
   a(i) = 1./i
   b(i) = a(i-1)+a(i)
ENDDO

What's wrong ??
What can be detected: Conflicts

Assure: Error Types

Write-Read conflicts

```fortran
!$OMP PARALLEL DO
DO i=1,N
   a = b+c(i)
   d(i) = a+e(i)
```

The 2 statements inside the loop have to be executed in that (Write-Read) order, which is not guaranteed in a multithreaded run (a is shared by default).

Repair: private(a)
Assure: Error Types

Read-Write conflicts

!$OMP PARALLEL DO
DO i=1,N
   d(i) = a+e(i)
   a = b+c(i)

Repair: private(a)

Assure: Error Types

Write-Write conflicts

!$OMP PARALLEL DO
DO i=1,N
   a = b+c(i)

Repair: private(a)
Private symbol, used outside loop

```fortran
!$OMP parallel do private(a)
DO i=1,N
    a = c(i)
    d(i) = a*a
ENDDO
PRINT*, a
```

Repair: `lastprivate(a)`

Uninitialized private

```fortran
firstiter = .TRUE.

!$OMP parallel do private(firstiter)
DO i=1,N
    IF( firstiter ) THEN ...
ENDDO
```

Repair: `firstprivate(firstiter)`
Assureview: Displays

Main

- Main error list
  - Clickable button for each error
    Click to get precise diagnostics
  - Overview chart showing statistics of bugs, different severities

- Call Tree
Assureview: Displays

Reading the diagnostics

Click the "+" buttons to get into the diagnostics

Finally the code sections are shown containing the error locations, (source and sink), both clearly marked.

AssureView: Displays

Inside code windows

- Show Search: normal string search menu
- Show Stack: show the calling sequence for arriving at the location.
Assureview: Displays

Other buttons

View
- Select display of the error list

Search
- Normal search menu, inside error list

Print
- Self explaining

Assureview: Displays

Preferences
- Miscellaneous settings. In particular:
  - source code locations ("finding files")

Reorder
- .. error list by different criteria
Thanks for your attention!