Disclaimer

- My opinions may not be my real opinions. They are certainly not anybody else's opinions.
- I may not agree with my comments after this panel
- I will ignore the questions of this panel

Software Challenges

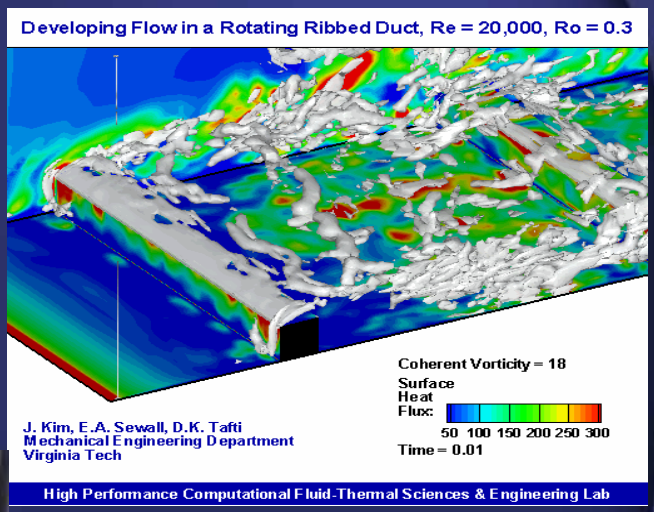
- Existing code must scale
  - Hundreds of thousands of processes/threads
  - Exploit variety of accelerators:
    - FPGAs, Cell Processors, GPUs, etc.
  - Optimizations across and within nodes
    - Addressing NUMA characteristics
- Parallel I/O
  - Language support, good strategies.
-

Case Study: MPI and OpenMP in a CFD Code

GenIDLEST (Tafti, VT)

- Fluid Dynamics Application written in MPI and OpenMP
- On a Sun Fire X4600 8 CPUs, each with 2-core Opteron 885 2.6GHz SMP running Linux
- OpenMP version of the code (8 threads) runs significantly slower than MPI (8 processes)… WHY???

Large Eddy Application



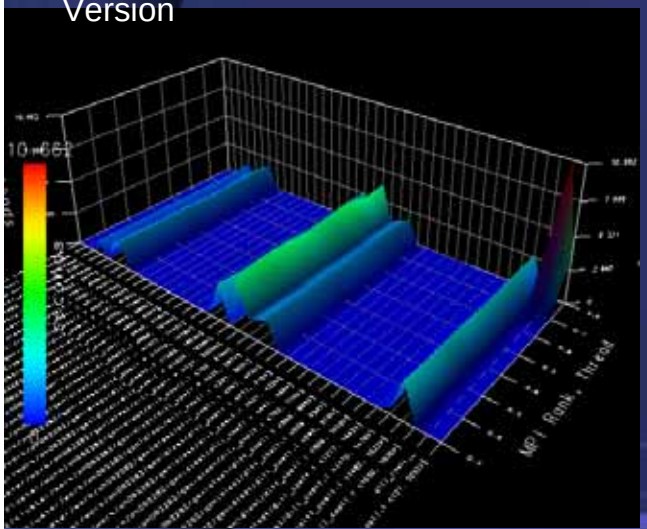
The Sun Fire X4600 16-core system

OpenUH Compiler

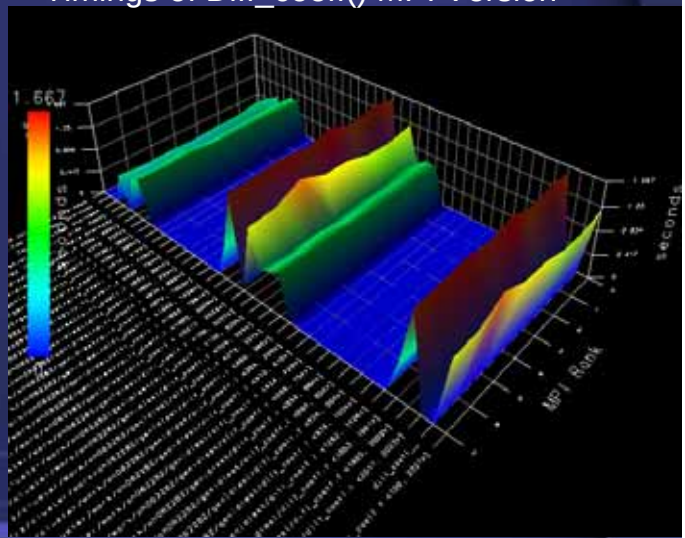
Understanding Why

The procedure Diff_coeff() was responsible for 20% of the execution time. We found that this procedure was 2.3 times slower than the MPI code version.

Timings of Diff_coeff() OpenMP Version



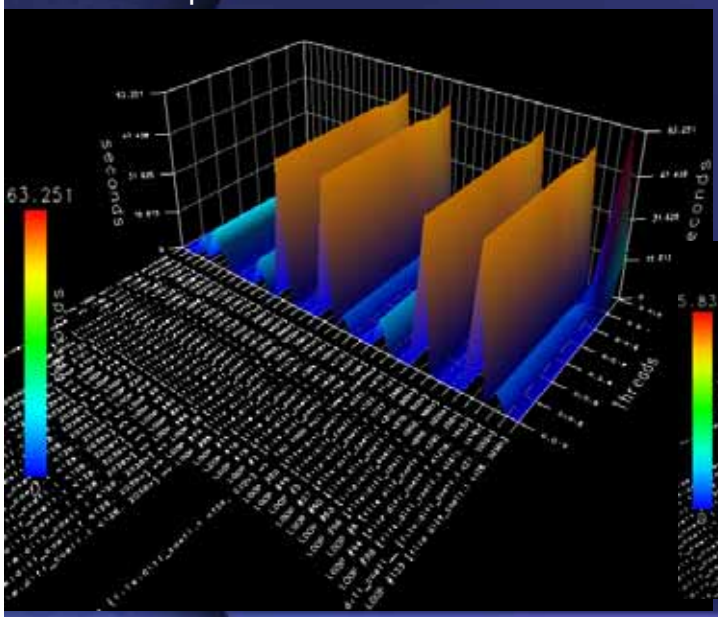
Timings of Diff_coeff() MPI Version



Note: Graph scales are different

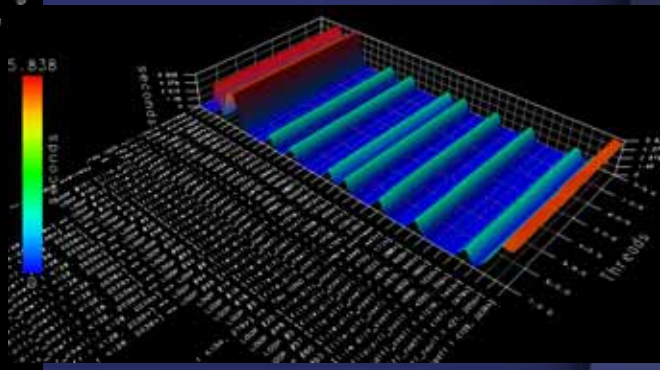
Understanding Why (cont.)

OpenMP version



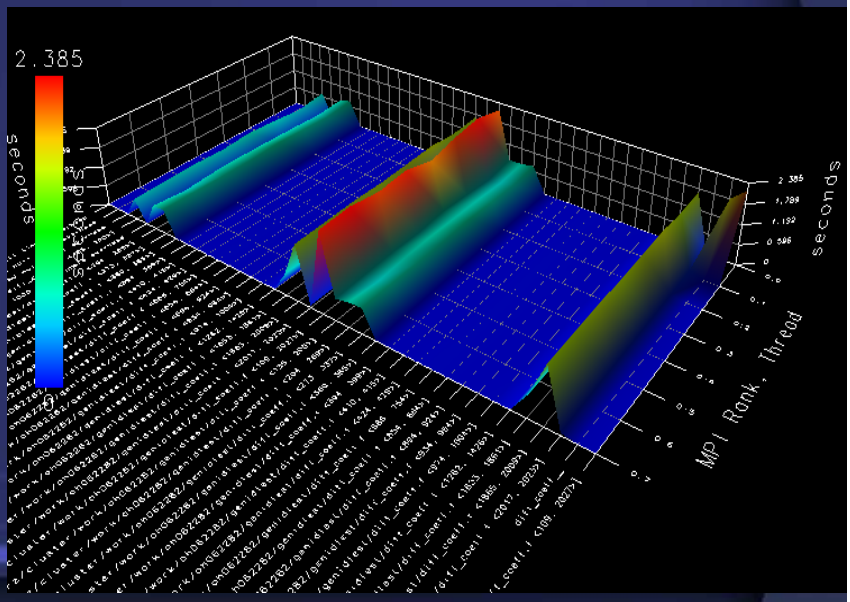
In the SGI Altix 3700 we observed the same difference between MPI and OpenMP versions.

MPI version



Optimizations for Diff_coef()

OpenMP version of Diff_coeff after optimizations



- We made sure the shared data structures were initialized to use “first touch”.
- We privatized most of the shared data of the procedure.
- We eliminated dynamic allocation of shared data.
- After these optimizations, the OpenMP version of Diff_coef() Improved by a factor of 1.87 times.

Overall, performance of OpenMP version nearly at MPI level

Lessons Learned

- Most performance problems were related to (lack of) (data) locality
- An expert in one parallel programming model may fail to get high levels of performance with another model
- We need a single productive programming model
  - (and very good tools)

Let's Call it Extended OpenMP

- Specification of data and computation locality is critical now and likely more so in future
  - We have to make it easy for application developer to specify this
  - At a suitable level of abstraction
  - both implicit and explicit strategies might work
- Can we provide these features in future OpenMP and retain its benefits?
  - incremental development
  - Compatible with sequential code
  - Productive programming

Example: SGI OpenMP Extensions

- SGI extensions to specify data distributions
- Basic mode *allocates pages to memory* on nodes; saved where “most of data” is needed
- This is inaccurate, but it is compatible with program translation on page-based system
- Alternate mode *allocates data to processors* in HPF style
- This is accurate, but it destroys illusion of shared memory and is harder to compile

Extended OpenMP Constructs

- This approach requires user to
  - specify data distribution explicitly
  - specify locus of thread execution
  - load balancing problems must be addressed

```
!$SGI     DISTRIBUTE array ( CYCLIC (1) )
!$OMP     PARALLEL DO PRIVATE ( i , active)
!$OMP& SHARED (  level )
!$SGI+    AFFINITY (i) = DATA ( array ( i ) )
           DO i = 1,  max
              IF (   array ( i )  >= 1) then
                 active = ….
                 CALL solve ( active, level, …)
              END IF
           END DO
```

Role of Compiler

- If we want productive programming, we should not attempt to “eliminate compiler”
- But we should work hard to improve our compiler technology (including dynamic)
  - And make it easy for the user to specify programs with a high degree of locality
- Compilers should be better integrated
  - give information on translation to users and tools
  - Integrate with static and dynamic tools

Parallel Data Flow Analysis: Motivation

```
int main()
{
 double a[N];
 double x = 3.1415;
 double y=3.1415926;
 double f1=2.3132;
 ...
#pragma omp parallel
 {
 #pragma omp single
  {
    x = x * y/f1 * k;
  }
#pragma omp for reduction(+:z)
for(j=0; j<1000; j++)
  for(i=0; i<N; i++)
  {
   a[i]= a[i] * x*y/f1;
   z = z + a[i] + x*y/f1 ;
  }
  printf("results: z = %d\n", z);
 }

}
```

(a) An OpenMP program

Compiler flags	-O3	-O3 –mp3
PRE-example	7.42	46.8
NAS FT	18.45	26.17
NAS UA	130.31	220.15

Why the different performance?

OpenMP and Compiler Optimizations

- Most compilers perform optimizations after OpenMP constructs have been lowered
  - Limits traditional optimizations
  - Misses opportunities for high level optimizations

```
#pragma omp parallel
{
#pragma omp single
  {
    k = 1 ;
  }
  if ( k==1) . . .
}
```

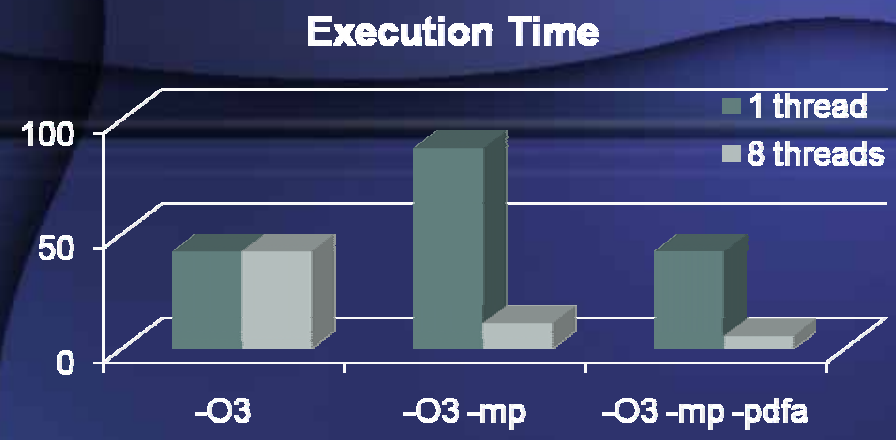
K==1? Yes

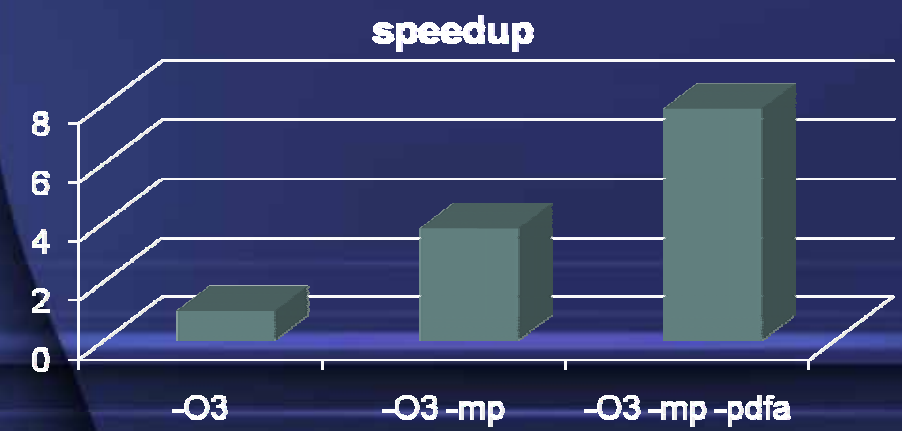
(a) An OpenMP program with a single construct

```
mpsp_status =
ompc_single(ompv_temp_gtid ) ;
i f (mpsp_status == 1)
{
  k = 1 ;
}
ompc_end_single ( ompv_temp_gtid );
if ( k==1) . . .
```

K==1? Unkown

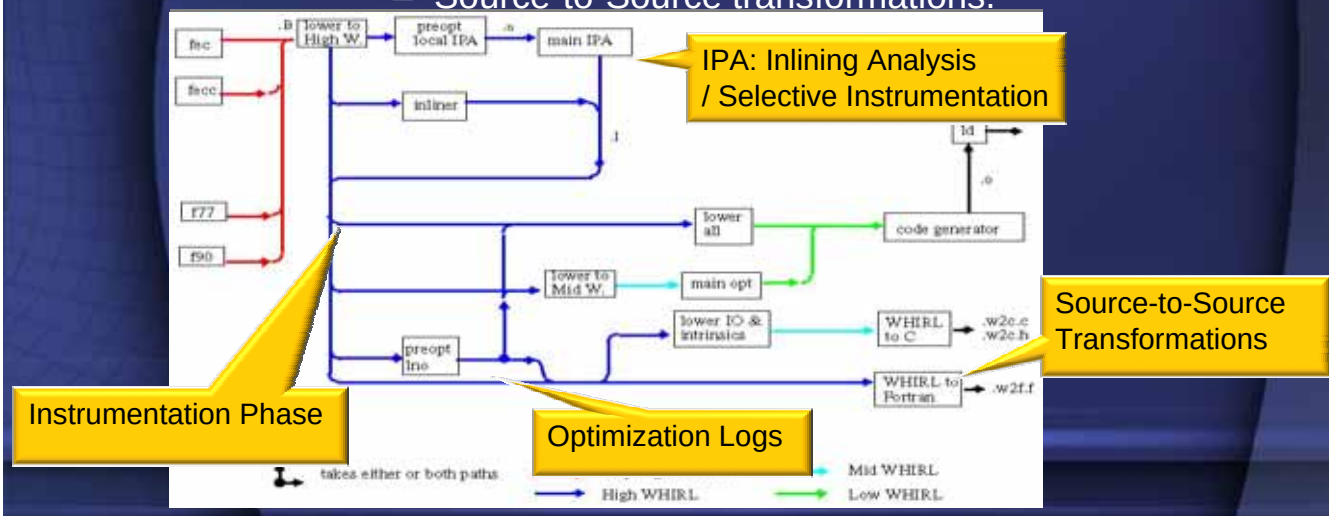
(b) The corresponding compiler translated threaded code





Compiler Support for Performance Analysis

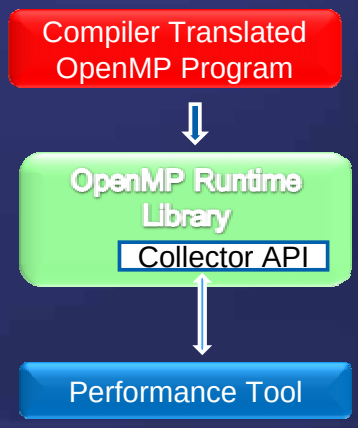
- Automatic instrumentation:
  - Improve instrumentation via IPA cost vector analysis.
- Saved compiler optimization logs:
  - Interpret / understand performance data.
  - Source-to-Source transformations.



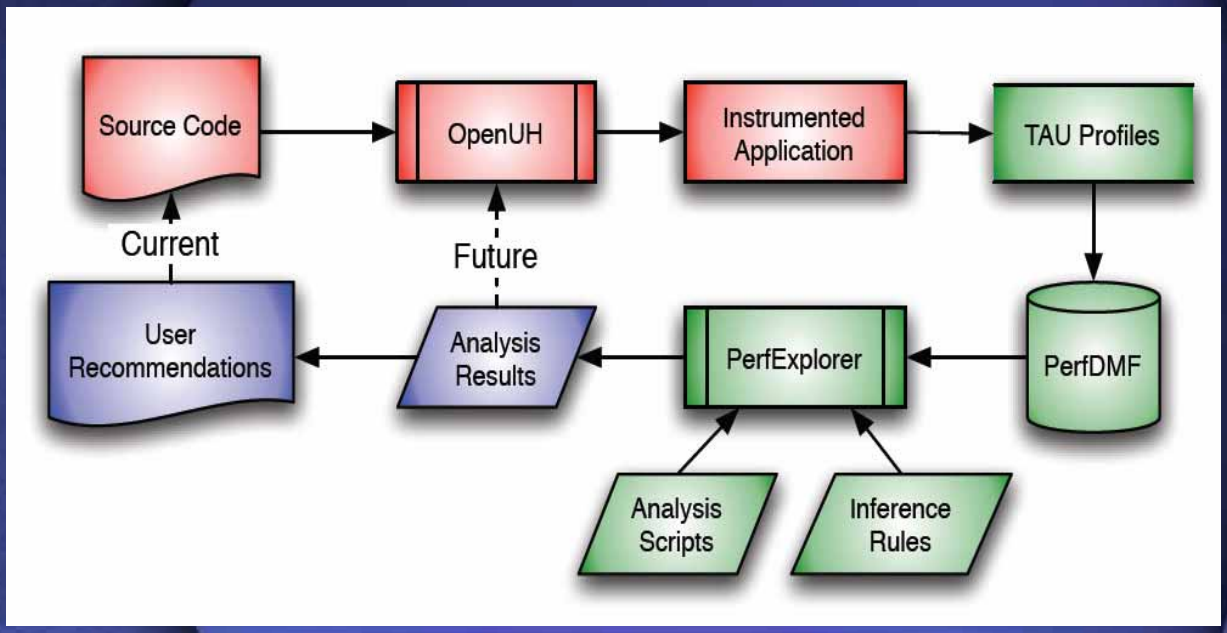
The OpenUH Compiler

Performance Monitoring Interface

- OpenMP ARB sanctioned performance monitoring interface for OpenMP
- Performance tools communicate with OpenMP runtime library through collector interface
- Designed to support statistical sampling
- Support tracing with extensions



Automating the Process



SC'08 paper: "Capturing Performance Knowledge for Automated Analysis"

Compiler Tools

- There is potential to create tools from compilers to address issues needed for high performances.
- More integration with performance tools needed and support for auto tuning.
- Tools should be able to "summarize" information and capture "expert knowledge"
- Compiler analyses need to support parallel models.