Design and Prototype of

Portability in this regard requires the de nition of the interface semantics and how information is to be accessed.

Allow exibility in how the interface is applied.

Since OpenMP compilers may implement OpenMP directives differently, including variations in runtime library operation, the performance interface should not constrain how it is used.

While our study focuses mainly on the instrumentation interface, as that is where events are monitored and the operational state is queried, clearly the type of performance measurement will determine the scope of analyses possible. Ideally, the exibility of the interface will support multiple measurement capabilities.

In this way, performance observation can be targeted at the levvs)8080-1.2983(get)285f628582 vl4(this)Tjicatures.553 32.72 (ation)

```
!$OMP INST [INIT | FINALIZE | ON | OFF]
```

must

3.9 Implementation Issues

It is clear that an interface for performance measurements must be very

The `pomp_begin` **and** `pomp_end` **routines would have an additional** `void*` **typed second**

out within the KOJAK project [11] and is a part of the ESPRIT working group APART [1].

EXPERT analyzes the performance behavior along three dimensions: *performance problem category*, *dynamic call tree position*, and *code location*. Each of the analyzed dimensions is organized in a hierarchy. Performance problems are organized from more general ("*There is an MPI related problem*") to very speci c ones ("*Messages sent in wrong order*"). The dynamic call tree is a natural hierarchy showing calling stack relationships. Finally, the location dimension represents the hierarchical hardware and software architecture of SMP clusters consisting of the levels machine, node, process, and thread.

The range of performance problems known to EXPERT are not hard-coded into the tool but are provided as a collection of *performance property speci cations*. This makes EXPERT extensible and very exible. A performance property speci cation consists of

- a compound event (i.e., an event pattern describing the nature of the performance problem),
- instructions to calculate the so-called *severity* of the property, determining its in uence on the performance of the analyzed application,

itsa(ent)Tj 27.326 0 Td (performance)Tj 52.7299 0 Td (propEXty)Tj 33.4324 0 FEER(T,)Tj 6.27056 0 Td (and)Tj

4.3 Integration into TAU

The TAU performance system [13] provides robust technology for performance instrumentation, measurement, and analysis for complex parallel palel cutegrations

Figure 2: EXPERT Performance Analysis of OpenMP/MPI Weather Forecasting Application Instrumented with OPARI

5.1 Weather Forecasting

The REMO weather forecast application from the DKRZ (Deutsches Klima Rechenzentrum, Germany) is an excellent testcase for the performance API. The code is instrumented using OPARI for OpenMP events and the MPI profiling library for MPI events. The measurement system uses the EPILOG

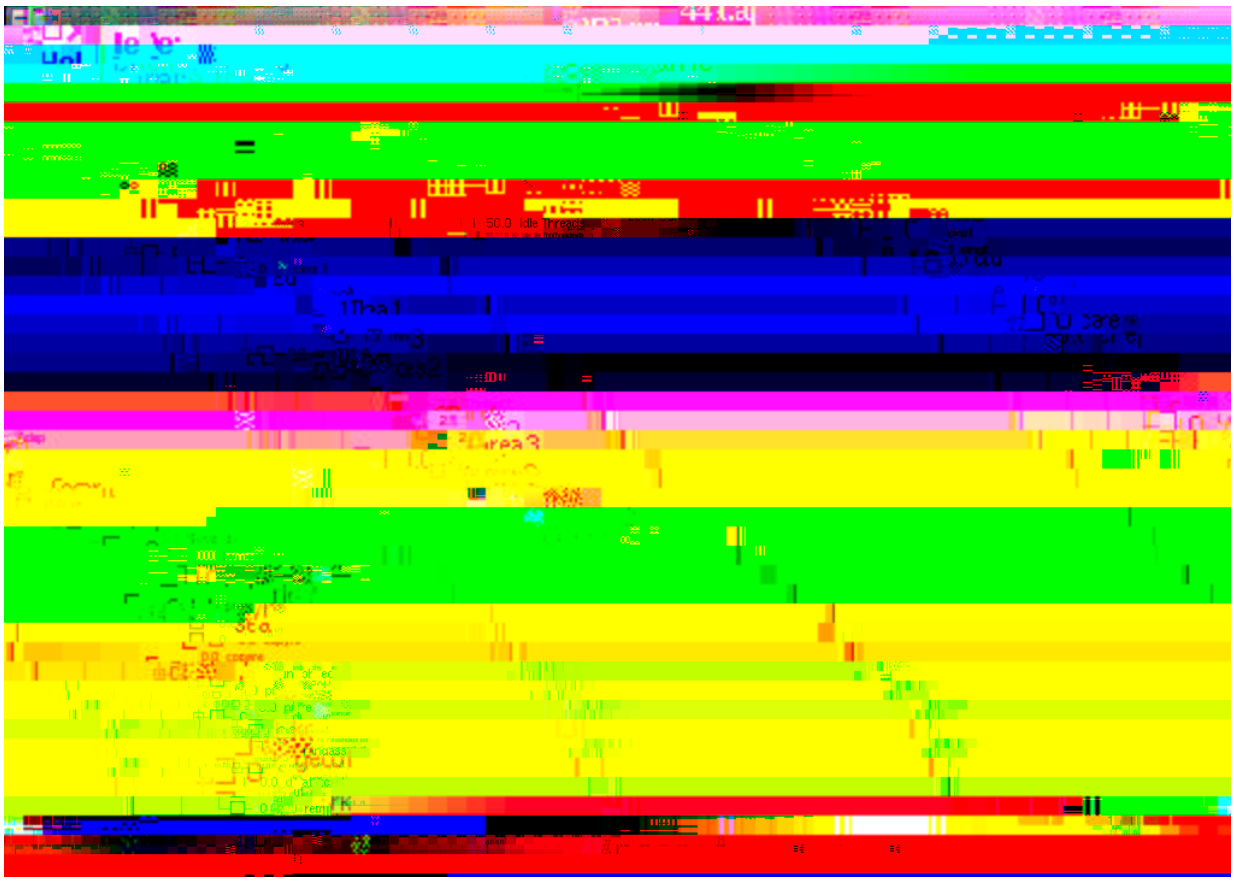


Figure 3: Barrier Performance Analysis of REMO

The right view shows the distribution of idle times across the different threads. Here all values refer to the selection in the left neighbor, so the sum of all values correspond to the 73.9% from the middle view. Of course, only the slave threads have idle times, the master thread shows always 0.0%.

Figure 3 refers to the property "OpenMP Barrier." The call tree shows that nearly all barrier time is spent on an implicit barrier (`!$omp ibarrier`) belonging to a parallel do (`!$omp do`). The distribution of overhead across the different threads refer to the

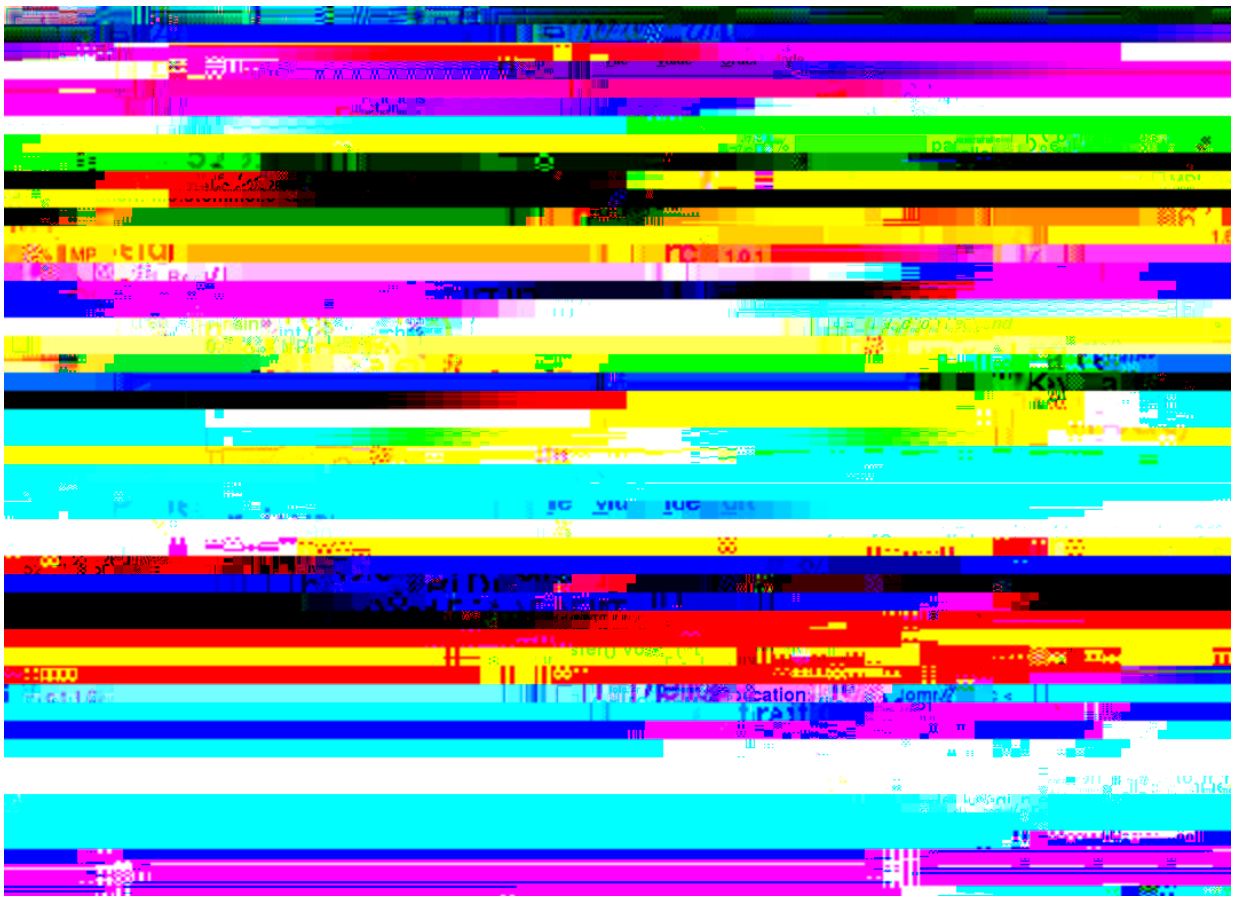
Original code block

```
#pragma omp for schedule(static) Oeduction(+: diff) private(j)
                    (a1,a2,a3,a4,a5)
  for(i=i1;i<=i2;i++) {
      for(j=j1;j<=j2;j++){
          new_psi[i][j]=a1*psi[i+1][j]
```

Code

```
pomp_for_enter(&omp_rd_2);
#line 252 "stommel.c"
#pragma omp for schedule(static) Oeduction(+: diff) private(j)
                    (a1,a2,a3,a4,a5) nowait
  for(i=i1;i<=i2;i++) {
      for(j=j1;j<=j2;j++){
          new_psi[i][j]=a1*psi[i+1][j]
```

Table 4:



the API also offers a target for OpenMP compilers to generate POMP calls that can both access internal, compiler-speci c