```
Analysis and Anomaly Detection; working discussion group
------------------------------

Marty
Kevin
Allen
Nathan
Dorian
Bernd
Todd
Stephane
Gabriel
Shirley
John
Luiz
Jesus
Heidi
David

Marty: anomaly detection
- is there something in the data that the user can address?
        - is there work to address this issue?
        - it's hard
        - Kojak, etc.

Bernd: current state - finite set of known problems which are searched for
Bernd: what makes an anomaly?
Jesus: anomaly doesn't lead to the true cause, just the symptoms
John: what are the useful places to target - idle time
- group in agreement.
Bernd/Marty - need distinguish between primary symptom and actual cause
- how do we move in that direction?
Allen: the tools are blind - how can we get them more information to restrict
the problem set
        root cause analysis at the application level:
                - master slave, bag of tasks, divide and conquer, etc.
                - what are the known anomalies?
                - what are the solutions?
                        - but the solutions have to make sense to the naive developer.
Luiz: can we define high level computational models?
Allen: Poirot - does analysis based on computational model definitions.
John: different metrics of success for each application.
Marty: is the hardware being used efficiently?
Todd: users want to know how many timesteps per measure of time are being
computed
Marty: user tells you it's a timestep based code, and you can look for the
pattern, or at least know about it
Luiz: higher level question: can we detect the computational models?
Heidi: two types of users - one who knows their code, and one who does not.
John: we can handle that - callpath profiling shows you where the time is
spent, and differential profiling can show scaling problems.  Then you may also
need time axis information.  We should be able to detect busy wait and non-busy
wait.
```

Marty: it's hard to imagine how the user can help us. We can see what the runtime behavior is, but how do we map that to the user's model of what is supposed to happen?
Allen: We can do that. We can map performance behavior to the application semantics. Phases for example. Does the application behave regularly? Do you expect it to?
Marty: users have an expectation, and often the code doesn't behave that way, and that is the cause of the problem.
Allen: porting example - runs well on one machine, doesn't on another.
John: that's where differential analysis comes in handy. You can compare one to the other, and see where the inefficiencies are.
Marty: and what's a cache miss? and is that a bad number?
Bernd: you can at least narrow the problem down to memory, or network, etc.
Heidi: how can we go top down without user intervention? but bottom up, you at least have to try something.
Bernd: "PERFORMANCE PROPERTIES" no value judgment (good/bad) - that's up to interpretation.
Heidi: that's why we compare performance measurement to "peak" potential. "utilization" is the safe term.
Shirley: paper studying barnes-hut. If you don't scale appropriately, you are making it worse. the terms used weren't even appropriate. need to construct the application specific performance model first, then apply performance analysis to that.
Allen: modeling of sweep3d example: built from knowledge of the app, not tool data.
Todd: application model helps us understand the performance.
Marty: so, how do we query the users to extract this model?
Kevin: what's the base level of model knowledge needed?
Shirley: what's the base level of performance information to validate the model?
John: can it be done without user involvement? what in the performance data matters to you as a user?
Nathan: you can synthesize from base metrics some powerful metrics.
Allen: how do we encode and capture this type of intelligence?
Heidi: what is the user's performance goal? why are they using the tool? power? scaling? time to completion? collect data based on the goals.
Marty: usually it's time to solution, or scaling to a larger problem. they don't necessarily care about efficiency.
Luiz: suspicious of "time to solution".
Heidi: good to get a characterization of the application BEFORE the effort to optimize is undertaken.
John: good to compare 1 core to 8 cores, and see scaling, and make a plan for optimization.
Heidi: bias that more cores == faster solution.
consensus: there are lots of potential characteristics for performance.

Break

Marty: important characterization: timesteps or similar. Can we get the user to outline their iteration boundaries?
Bernd: simple and more complex instrumentation options. some things should be optional.
Todd: need some way to indicate application progress. iterations, convergence, etc.
Bernd: can we define an instrumentation API that ALL TOOLS will support?

Heidi: users can get used to it if it is commonly available.
Todd/Allen: also support for application specific counters/timers (i.e. annotations) could we have a common syntax for supporting it.
Todd: progress loops and effort loops. progress == main iteration loop, and effort loops are somewhat undefined (variable amount of time).
Bernd: higher level - something that semantically makes sense to the user. Simple naming is useful for visualization, but for analysis, you need something that the tool can make sense of.
Marty: do we need the extra semantics?
Allen: we need to distinguish between users who know their code, and those who don't. The users who know their code can be more helpful.
Bernd: agree that a simple two level loop could be enough. global sync and effort step.
Allen: don't see the value of this.
Todd: what is it that the application developer cares about?
Heidi: how do we "market" this common annotation to the users? to new code development? to sophisticated users?
Marty: long lived codes will benefit from this when codes move from one machine to another, as this has been supported by sun, ibm, cray, etc.
Allen: ok, it's a good start, but there is more that the user can specify in annotation.
Bernd: but we should have a default implementation as well. the user should be trained that something useful can come from it
Nathan: why not just use tracing? Why do you need the annotation at all?

Shirley: interesting issue about rewriting applications for multicore.
Marty: users will expect the compiler to handle it.
Shirley: yeah, but even so, we need to measure how the compiler did. Also, we need to consider the special issues of petascale. especially petascale with multicore.
David: it depends on how memory bandwidth limited the applications are. Fast is important for everyone, and there is some flexibility on the application side for the future machines.
Bernd/Marty: The new economics is in the memory and interconnect, not in cores.
The group wandered into a hardware counter discussion at this point...
multiplexing v. multiple runs

Going back...
Marty: It's good to have a standard which was supported by the tools, it's not clear that the users care enough to instrument with the tool. Is this a task that this group wants to tackle?
Nathan: don't think this is a feature that our group (Rice) is interested in.
Marty: the question on the floor is that is it worth our time to support these annotations to guide the analysis and make the analysis more meaningful to the user's understanding of the application?
Vote: the group is split.
Shirley: could also be used to help validate the model, if the model is annotated.
David: should also engage with the component/module groups to help support this effort.
Jesus: we already do periodicity detection through FFT. We have other ways to detect periodic regions of code. It doesn't hurt to have the user annotation. But it needs to have as little semantics as possible to keep it simple.
Bernd: but the more semantics you have they are helpful.

Jesus: but the semantics are vague.

Summary: we (most of us) want some user input to help the analysis tools.
Bernd: I am willing to run with it if tool users think it is useful.
Marty: we never talked about the anomalies we want to detect. Shall we meet again?
What are the topics we can agree on and make progress on?

Decision: we will summarize tomorrow, and regroup.