Knowledge Engineering for Model-based Parallel Performance Diagnosis

***Li Li* and *Allen D. Malony*,** *{lili,malony}@cs.uoregon.edu*
http://www.cs.uoregon.edu/research/tau
Computer and Information Science Department, University of Oregon, Eugene, OR

Abstract

Scientific parallel programs often undergo significant performance tuning before meeting performance expectation. Performance tuning naturally involves a diagnosis process - locating performance bugs that make a program inefficient and explaining them in terms of high-level program design. We present a systematic approach to generating performance knowledge for automatically diagnosing parallel programs. Our approach exploits program semantics and parallelism embedded in computational models to search and explain bugs. We first describe how to extract performance knowledge from parallel models. Second, we represent the knowledge in such a way that diagnosis can be carried out in an automatic manner. We then present *Hercule* - a prototype automatic performance diagnosis system - and our evaluation system for validating diagnosis results. Our experience diagnosing Master-Worker programs show that model-based performance knowledge can provide effective guidance for locating and explaining performance bugs at a high level of program abstraction.

Introduction

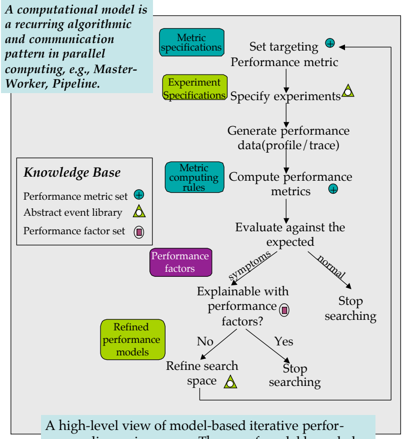
Two observations of existing important performance measurement and analysis tools particularly motivate our work:

• The performance feedbacks provided by the tools tend to be descriptive information about parallel program execution at low level of program abstraction.

• The design of performance experiments, examining performance data, and evaluating performance against the expected to identify performance bugs are not well automated and not necessarily guided by a diagnosis strategy.

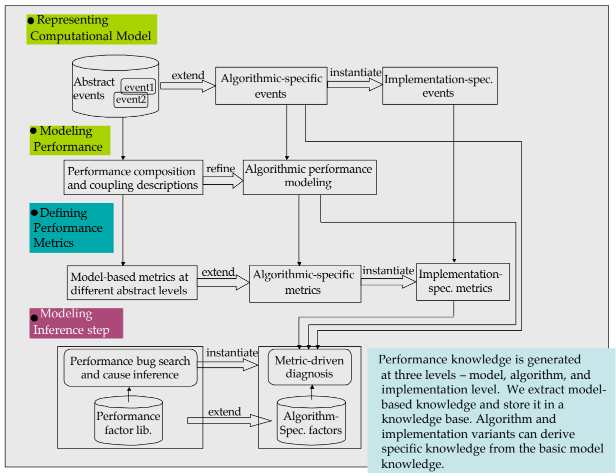
Our approach to performance diagnosis is based on parallel computational models.

A computational model is a recurring algorithmic and communication pattern in parallel computing, e.g., Master-Worker, Pipeline.



A high-level view of model-based iterative performance diagnosis process. The use of model knowledge is annotated on the process steps.

Generation of Model-based Performance Knowledge


● Representing Computational Model
Abstract events
event1
event2
extend
Algorithmic-specific events
instantiate
Implementation-spec. events
● Modeling Performance
Performance composition and coupling descriptions
refine
Algorithmic performance modeling
● Defining Performance Metrics
Model-based metrics at different abstract levels
extend
Algorithmic-specific metrics
instantiate
Implementation-spec. metrics
● Modeling Inference step
Performance bug search and cause inference
instantiate
Metric-driven diagnosis
Performance factor lib.
extend
Algorithm-Spec. factors

Performance knowledge is generated at three levels – model, algorithm, and implementation level. We extract model-based knowledge and store it in a knowledge base. Algorithm and implementation variants can derive specific knowledge from the basic model knowledge.

Representation of Performance Knowledge

How do we represent performance knowledge in a knowledge base so that diagnosis process can be performed with minimum user intervention and algorithm- and implementation-specific performance information can be readily derived and incorporated into the system?

we create an ***inference tree*** that represents our bottom-up performance diagnosis approach and formalizes a structured knowledge invocation process.

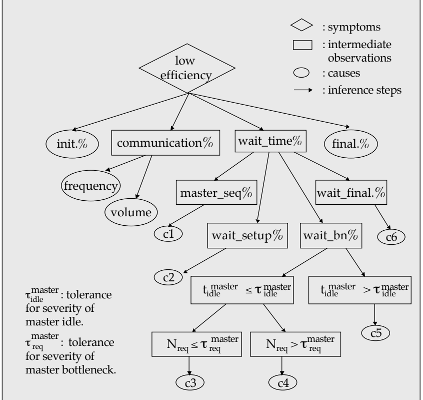
•The root of the tree represents the symptom that we are going to diagnose.

•Branch nodes represent *intermediate observations* that we have achieved so far and need further performance evidences to explain,.

•Leaf nodes represent an explanation of the root symptom in terms of high-level performance factors.

•Readily incorporate knowledge generated from algorithm variants through adding branches at appropriate tree levels.

•An example inference tree of a Master-Worker model is presented in the figure below.


: symptoms
: intermediate observations
: causes
: inference steps
low efficiency
init.%
communication%
wait_time%
final.%
frequency
volume
master_seq%
wait_final.%
c1
wait_setup%
wait_bn%
c6
c2
$t_{idle}^{master} \le \tau_{idle}^{master}$
$t_{idle}^{master} > \tau_{idle}^{master}$
c5
$N_{req} \le \tau_{req}^{master}$
$N_{req} > \tau_{req}^{master}$
c3
c4

τ_{idle}^{master} : tolerance for severity of master idle.

τ_{req}^{master} : tolerance for severity of master bottleneck.

c1: Sequential initialization and finalization at the master accounts for lost cycles of the worker.

c2: master setup task overhead is significant, which means the speed of master processing request is pretty slow.

c3: During the execution the master ware rarely idle. The worker spent quite time waiting in master bottleneck, but only a few worker requests got stuck in the bottlenecks. The situation indicates that master processing request speed is slow. In other words, task setup cost is expensive relative to task computation cost.

c4: During this execution, the master was rarely idle, which implies that there is little room for rescheduling to improve performance. The facts that the worker spent quite time waiting in master bottlenecks and a number of worker requests got stuck at the master imply that the amount of workers used exceeds the needed, given the master processing speed and the input problem size.

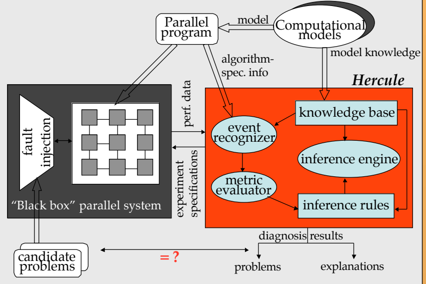
c5: Master idle time is significant. There is room for improving performance by rescheduling. Adjusting task assignment order in a way that keeps the mater busy while avoiding bottlenecks.

c6: Time imbalance is significant. Try to normalize the last task finish time of all workers.

Conclusions

We describe a systematic approach to generating and representing performance knowledge for the purpose of automatic performance diagnosis. The methodology makes use of operation semantics and parallelism found in parallel computational models as a basis for performance bug search and explanation. Four major categories of knowledge are identified: abstract events that describe model behaviors, performance modeling, model-specific metrics, and performance cause inference steps. A knowledge base is engineered by inputting expert information in each category for parallel models of interest. The methodology also addresses the adaptability of knowledge generation to algorithm and implementation variants. One important objective of our work is to study how diagnosis knowledge is represented. In this work, we showed the use of *inference tree* for formalizing a structured knowledge invocation process for automatic knowledge diagnosis. We also demonstrate the use of prototype *Hercule* parallel performance diagnosis system on a representative programming paradigm, Master-Worker. Our preliminary results show that model-based performance knowledge provides effective guidance for locating and explaining performance bugs at a high level of program abstraction.

Hercule - A Prototype Automatic Performance Diagnosis System



• *Knowledge base* in *Hercule* consists of abstract event library, metric set, and performance factors for each individual computational model.

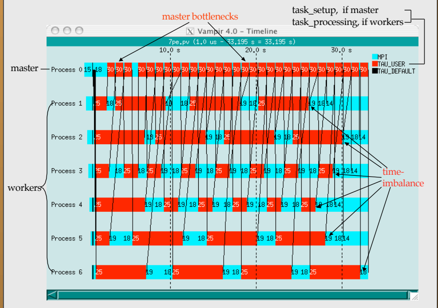
• *Event recognizer* and *metric evaluator* accept user-defined abstract event descriptions and performance metric definitions for algorithm-specific performance evaluation.

• *Inference engine* repeatedly fires rules with original and derived performance information until no more new facts can be produced, thereby realizing automatic performance reasoning.

• We use a *"black box" parallel system* to validate *Hercule* diagnosis results in a controlled manner. Candidate performance problems are injected to the system. *Hercule,* being only informed of the computational model the parallel program is patterned on, reaches diagnosis conclusions. We compare the conclusions against the problems introduced at the start to validate.

Experiments and Results

A problematic Master-Worker program run with 7 processes on a SMP cluster with 16 dual processor nodes connected through gigabyte switches.



Hercule diagnosis results output

```
dyna6-166:~/PerfDiagnosis/classes lili$ ./model_diag MW.clp
Begin diagnosing
... ...
Level 1 experiment - collect data for computing worker efficiencies.
__________________________________________________________________
Worker 3 is least utilized, whose efficiency is 0.385.
__________________________________________________________________
Level 2 experiment - collect data for computing initialization,
communication, finalization costs, and waiting (idle) time of worker 3.
__________________________________________________________________
Waiting time of worker 3 is significant.
__________________________________________________________________
Level 3 experiment - collect data for computing individual waiting
time fields.
__________________________________________________________________
Among lost cycles of worker 3, 14.831% is spent waiting for the last
worker to finish its computation (time imbalance).
__________________________________________________________________
Master processing time for assigning task to workers is significant
relative to average task processing time, which causes workers to
wait a while for task assignments. Among lost cycles of worker 3,
34.301% is spent waiting for master setting up task assignment.
__________________________________________________________________
Among lost cycles of worker 3, 39.227% is spent waiting for the
master to process other workers' requests in bottlenecks. This is
because master processing time for assigning task is expensive
relative to average task processing time, which causes some
workers to queue up waiting for task assignment.
__________________________________________________________________
```