PICS - a Performance-analysis-based Introspective Control System to Steer Parallel Applications

Yanhua Sun, Laxmikant V. Kalé

University of Illinois at Urbana-Champaign

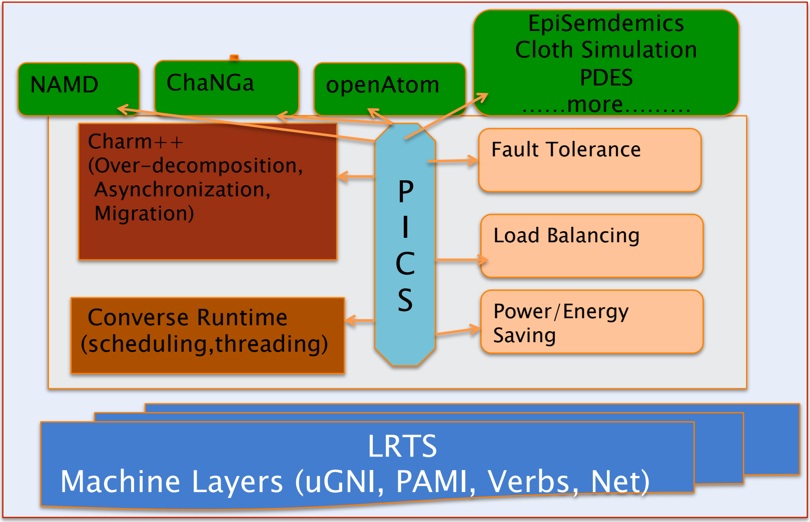
sun51@illinois.edu

November 26, 2014

Parallel Programming Laboratory @UIUC

- PPL : led by Professor Kalé since 1985 (30 years)
- Group of research staff, post-doc, graduate students, undergraduate (20+)
- Charm++ programming model and runtime system, real world applications (open source)
- 12 Charm++ workshops

Charm++



NAMD
ChaNGa
openAtom
EpiSemdemics
Cloth Simulation
PDES
......more.........
Charm++
(Over-decomposition,
Asynchronization,
Migration)
Converse Runtime
(scheduling,threading)
P
I
C
S
Fault Tolerance
Load Balancing
Power/Energy
Saving
LRTS
Machine Layers (uGNI, PAMI, Verbs, Net)

Goal : Productivity + Performance

- Asynchronous, message driven, over-decomposition programming model
- More control: mapping, load balancing, memory management, communication optimization
- Observability and controllability

Most important feature : load balancing

Why not a general scheme to enhance the adaptivity?

Goal : Productivity + Performance

- Asynchronous, message driven, over-decomposition programming model
- More control: mapping, load balancing, memory management, communication optimization
- Observability and controllability

Most important feature : load balancing

Why not a general scheme to enhance the adaptivity?
PICS : Control point centered introspective control system to steer applications and runtime system

Observation

Configurations of tunable parameters in the runtime system and applications significantly affect the performance.

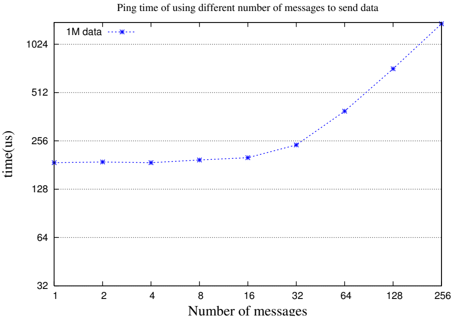


Figure: Data transfer without computation

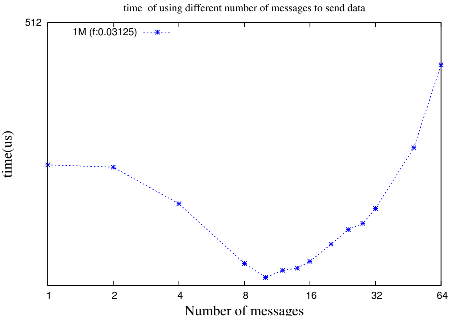


Figure: Data transfer with computation

Principle of Persistence

Things rarely change suddenly

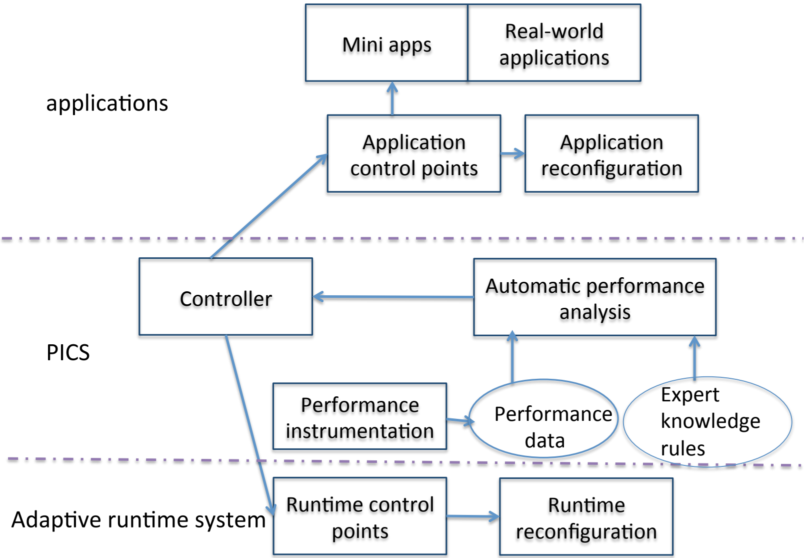
Principle of Persistence

Things rarely change suddenly

Principle of Persistence

Things rarely change suddenly

Overview of PICS framework



Control Points

Control points

Control points are tunable parameters for application and runtime to interact with control system. First proposed in Dooley's research.

1. Name, Values : default, min, max
2. Movement unit: $+1, \times 2$
3. Associated function, object, array
4. Effects, directions
    - Degree of parallelism
    - Grainsize
    - Priority
    - Memory usage
    - GPU load
    - Message size
    - Number of messages
    - other effects

Application and Control Points

Application

1. Application specific control points provided by users
2. Applications should be able to reconfigure to use new values

Runtime

1. Registered by runtime itself
2. Requires no change from applications
3. Affect all applications

Control points	Effects	Use Cases
sub-block size	parallelism, grain size	Jacobi, Wave, stencil code
parallel threshold	parallelism, overhead, grain size	state space search
stages in pipeline	number of messages, message size	pipeline collectives
algorithm selection	degree of parallelism, grain size	3D FFT decomposition (slab or pencil)
software cache size	memory usage, amount of communication	ChaNGa
ratio of GPU CPU load	computation, load balance	NAMD, ChaNGa

Observe Program Behaviors

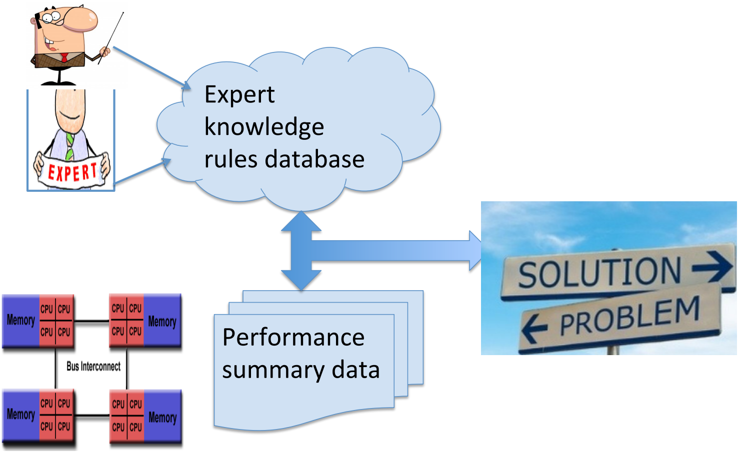
- Record all events
  - Events : begin idle, end idle
  - Functions: name, begin execution, end execution
  - Communication : message creation, size, source/destination
  - Hardware counters
- Module link, no source code modification
- Performance summary data

Automatically Analyze the Performance

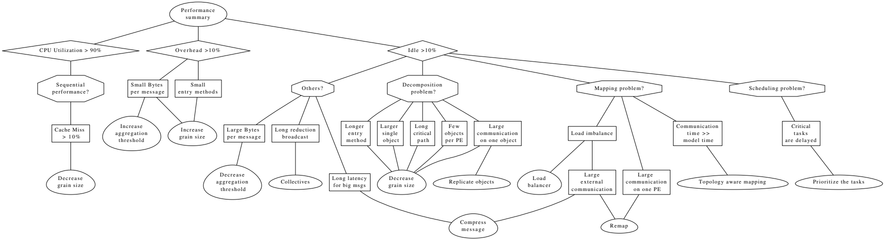
Many control points are registered. How to reduce the search space?

Automatically Analyze the Performance

Many control points are registered. How to reduce the search space? Performance analysis to identify program problems to narrow down the control points



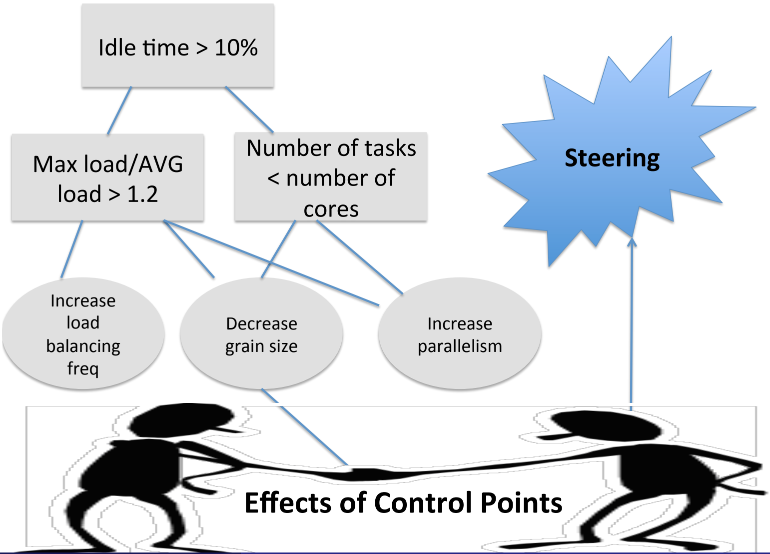
Decision Tree Based Performance Analysis



- Encoded in a plain text file
- Constructed at the beginning
- Dynamic learning new rules

Correlate Performance with Control Points

Traverse the tree using the performance summary results



Control System APIs

```
typedef struct ControlPoint_t
{
    char      name[30];
    enum      TP_DATATYPE datatype;
    double    defaultValue;
    double    currentValue;
    double    minValue;
    double    maxValue;
    double    bestValue;
    double    moveUnit;
    int       moveOP;
    int       effect;
    int       effectDirection;
    int       strategy;
    int       entryEP;
    int       objectID;
}ControlPoint;
```

APIs for applications

```
void registerControlPoint(ControlPoint *tp);

void startStep();
void endStep();

void startPhase(int phaseId);
void endPhase();

double getTunedParameter(const char *name, bool *valid);
```

Jacobi3d Performance Steering

- Control Points: sub-block size in each dimension
- Three control points
- Cache miss rate, high idle suggest decreases sub-block size
- Overhead

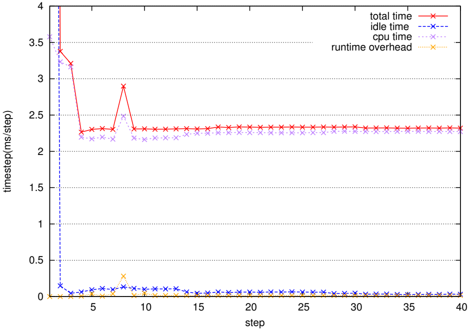


Figure: Jacobi3d performance steering on 64 cores for problem of 1024*1024*1024

Communication Bottleneck in ChaNGa

- Control points: number of mirrors
- Ratio of maximum communication per object to average

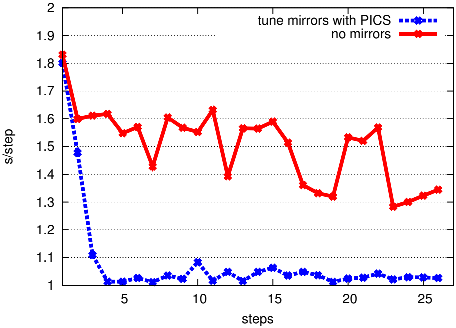


Figure: Time cost of calculating gravity for various mirrors and no mirror on 16k cores on Blue Gene/Q

Conclusion

- Application developers can provide hints to help optimize applications
- Automatic performance analysis helps guide performance steering
- Steering both runtime system and applications is important

http://charm.cs.illinois.edu
mailing list: charm@cs.illinois.edu