A Generic Adaptive Runtime Autotuning Framework

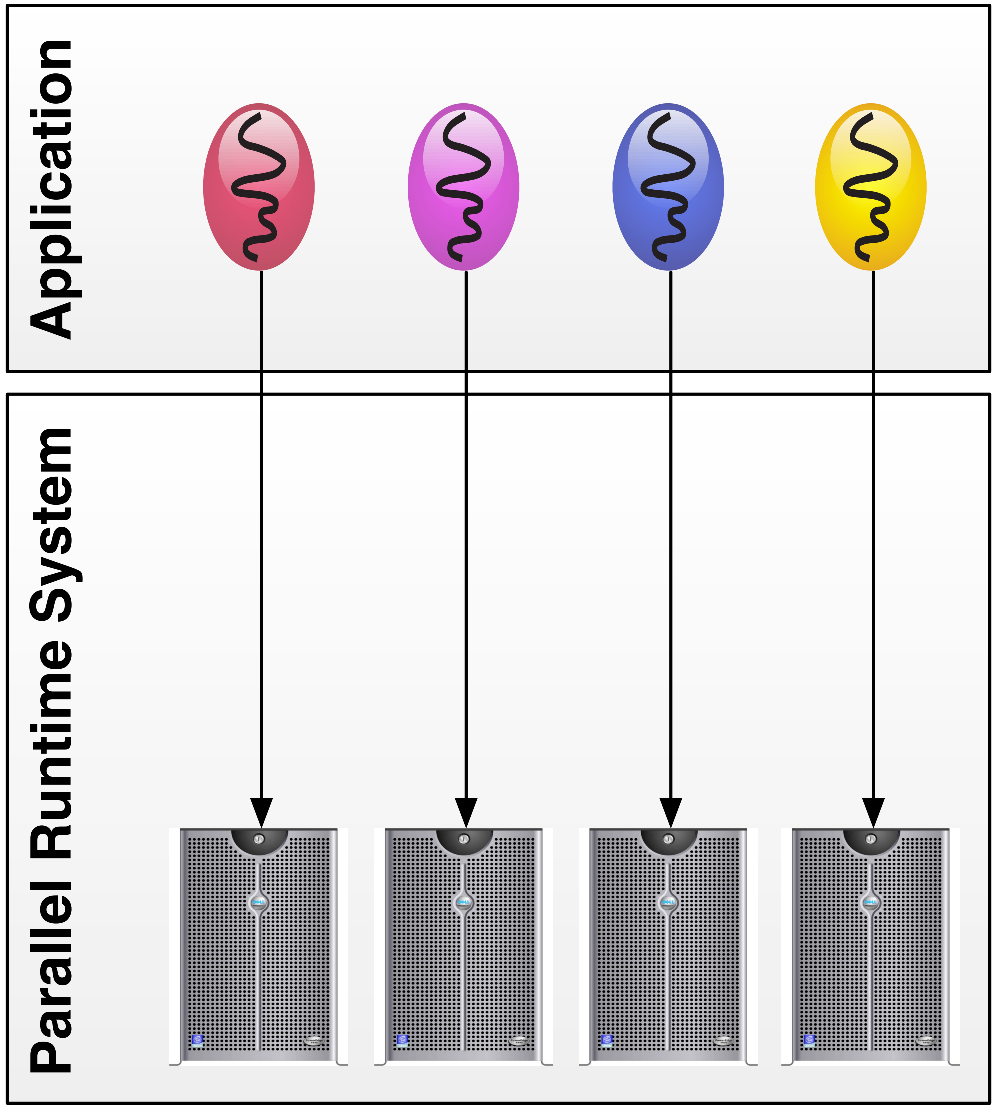
Isaac Dooley

7th Annual Workshop on Charm++ and its Applications

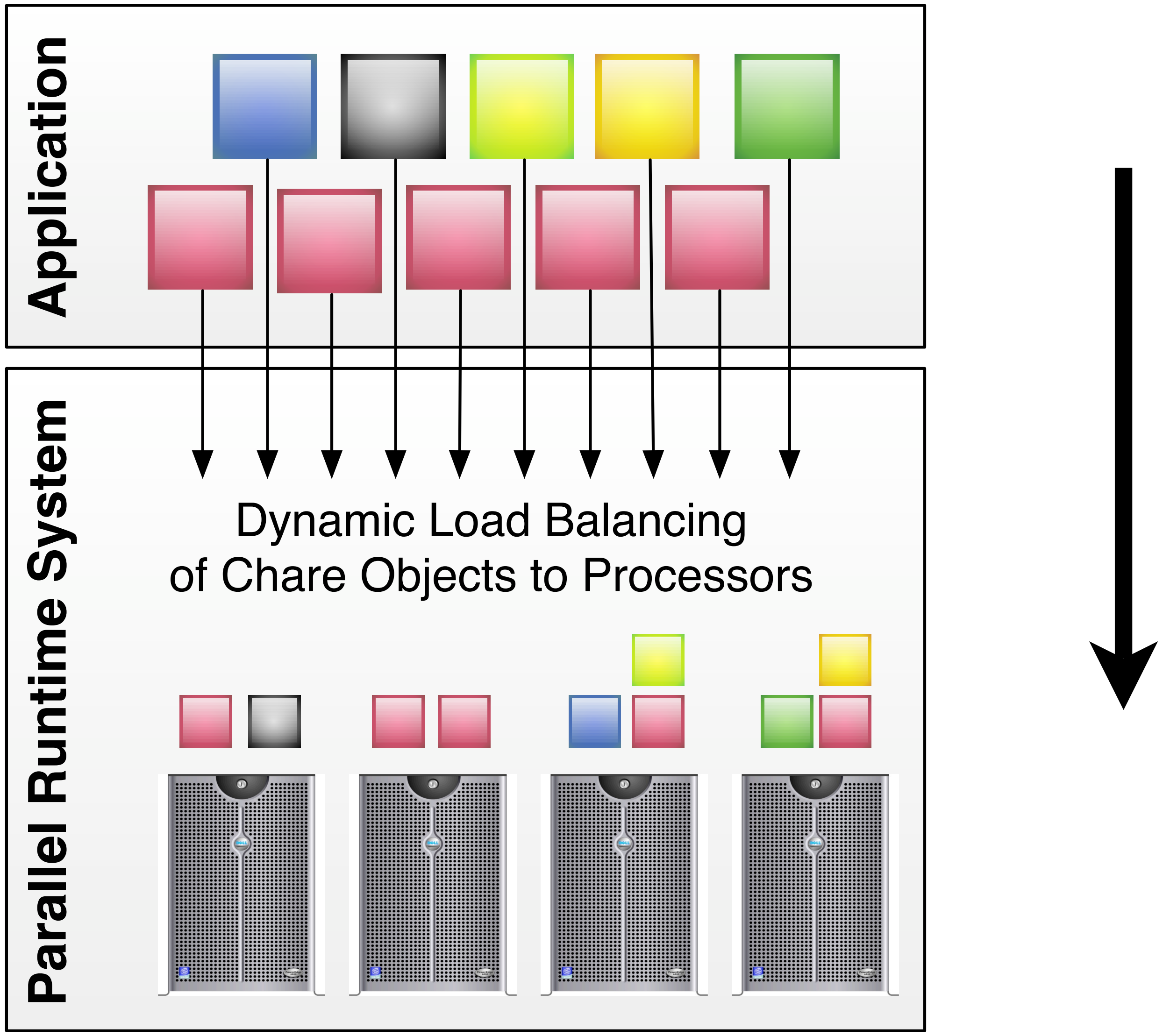
Thursday, April 16th, 2009

Existing Parallel Programming Models

MPI Model
One Thread Per Processor

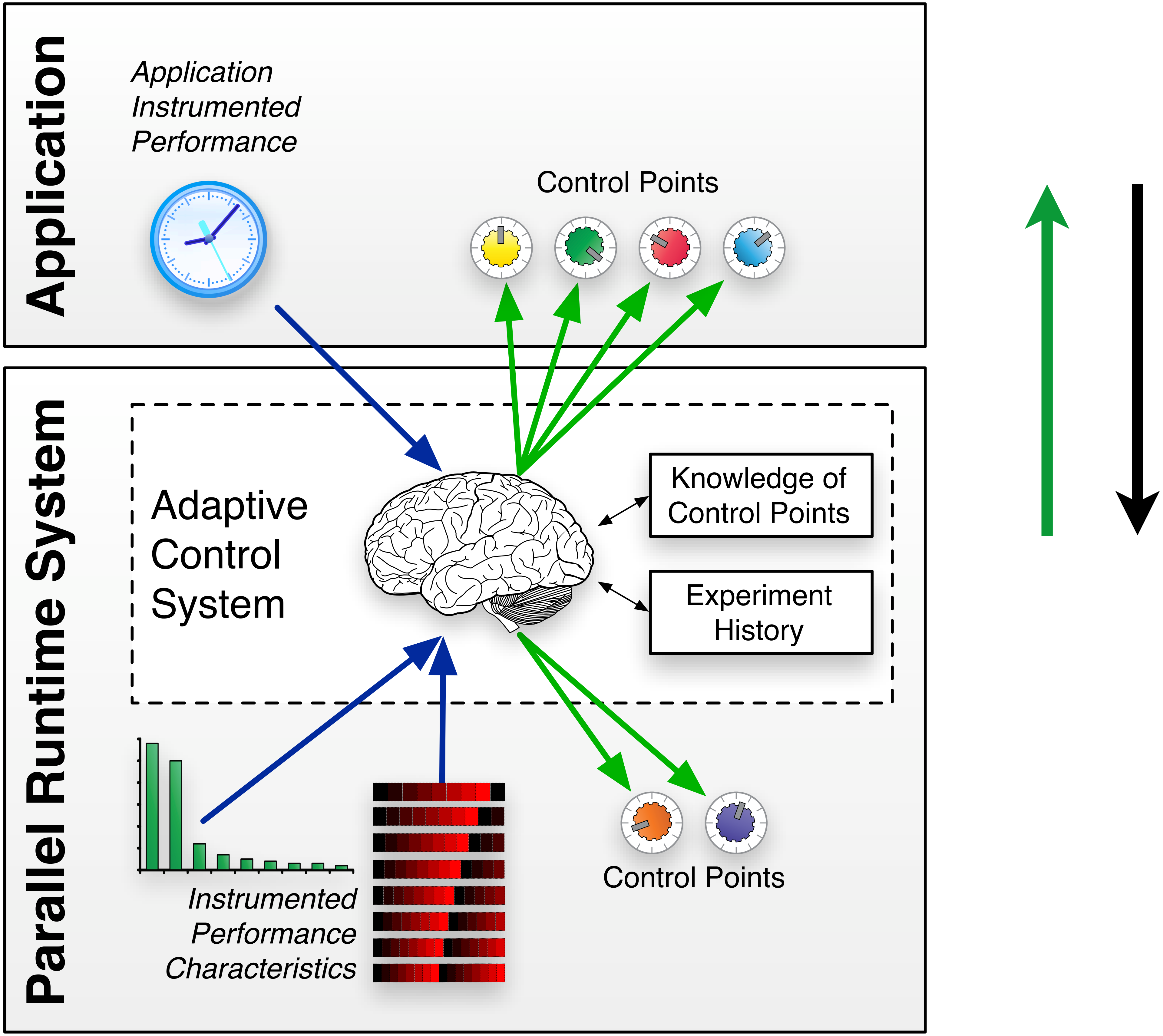


Charm++ Model
Overdecomposition



Dynamic Load Balancing
of Chare Objects to Processors

Runtime System Controls the Application


Application
Application Instrumented Performance
Control Points
Parallel Runtime System
Adaptive Control System
Knowledge of Control Points
Experiment History
Instrumented Performance Characteristics
Control Points

Intelligent Tuning

Measured Performance Metrics (Input to Controller)
Processor Utilization
Processor Overhead
Memory Utilization
Cache Performance
Application Decomposition Granularity
Communication Volume
Critical Path Profiling

Descriptive Categorizations for Application Behavior as Control Point Values are Increased
Task Decomposition Granularity
Task Scheduling Priorities
Degree of Pipeline Streaming
Memory Usage
Prefetch / Lookahead Distance

Control Point API

Application Exposes Control Point Values:

```
int controlPointValue = controlPoint("Control Point Name", 1, 50);
```

Application Specified Performance:

```
registerControlPointTiming(time);
```

Control Point Framework Instructs Application to adapt:

```
CkCallback myCallback (CkIndex_Main::controlPointChange(NULL),proxy);
registerControlPointChangeCallback(myCallback);
```

Describe Knowledge:

```
controlPointPriorityArray("Control Point Name", ArrayProxy);
controlPointPriorityEntry("Control Point Name", EntryMethod);
```

Use Cases

Adjust task/data granularity

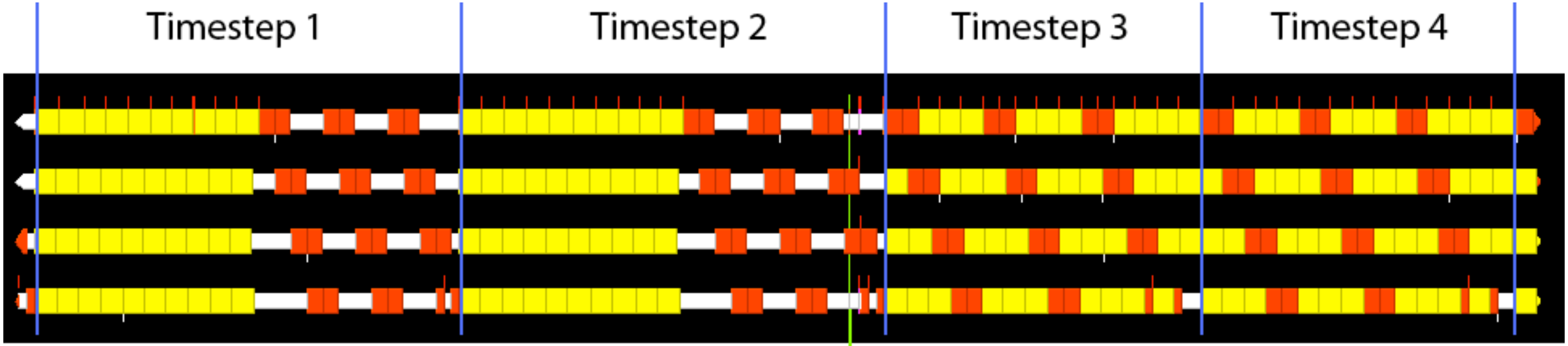
Adjust scheduling priorities

Adjust load balancing parameters

Choose algorithmic alternatives

Apply various communication optimizations

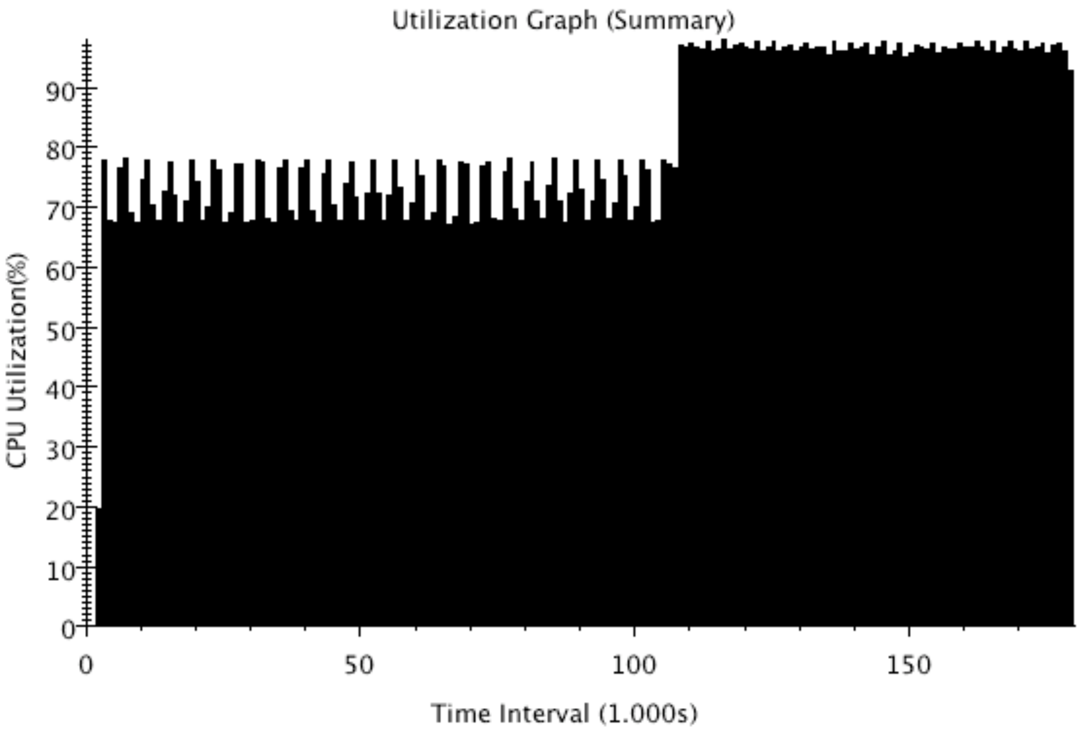
Tuning Critical Path Priorities



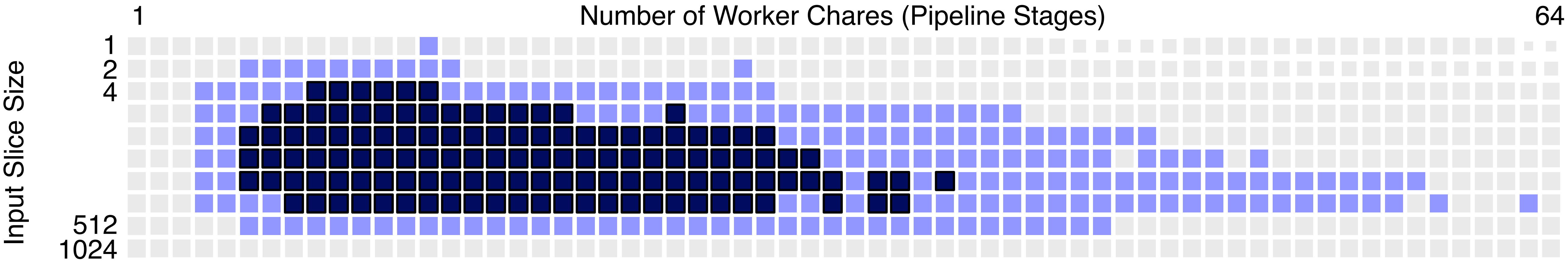
Same priorities for Yellow & Red

Control Point Framework increases priority for a control point related to the Red entry methods, because they are on the critical path

The new priorities improve overall performance



Control Point Configuration Space
Pipelined Filtering



Legend:

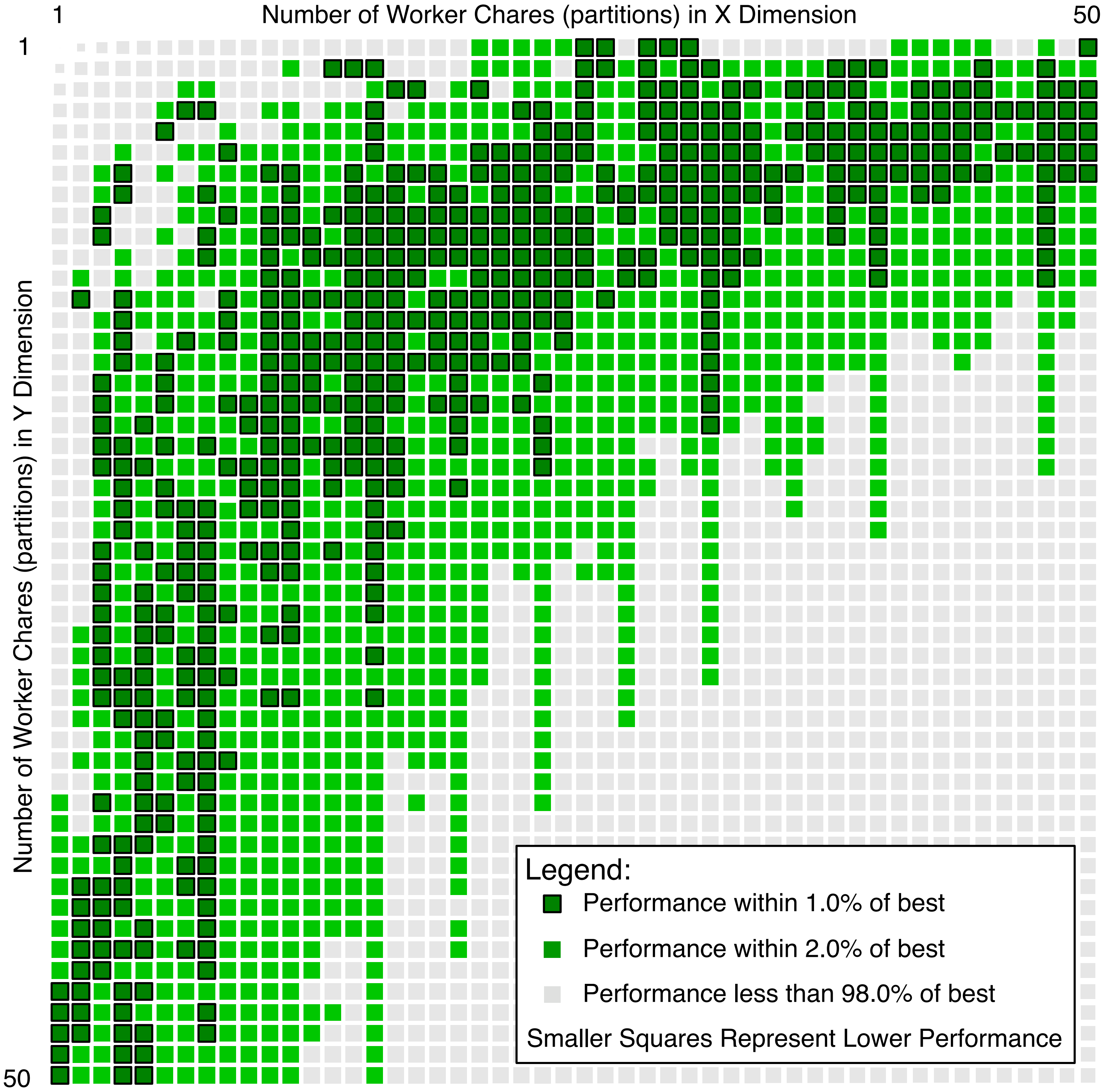
■ Performance within 1.0% of best

■ Performance within 2.0% of best

■ Performance less than 98.0% of best

Smaller Squares Represent Lower Performance

Control Point Configuration Space
2D Jacobi



Future Work

Improve critical path profiles.

Detect & fix more patterns of known performance problems.

Use with complicated applications & algorithms such as MD and LU.

Find appropriate ways to expose application knowledge.

Build an expert system combining all the patterns we discover.

The End

Questions?

Suggestions?

Isaac Dooley
idooley2@uiuc.edu