Online Specialization of Systems with Iridescent

Vaastav Anand (vaastav@mpi-sws.org)
Deepak Garg (dg@mpi-sws.org)
Antoine Kaufmann (antoinek@mpi-sws.org)

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

Modern cloud systems make trade-offs between flexibility and performance

Idea: Specialize and optimize code (by recompiling) at runtime!

Performance depends on 3 factors

Workload — Deployment Environment — Config Params

UNCONTROLLED UNCONTROLLED

Iridescent provides 4 building blocks

Spec. Points + Spec. Runtime + Explorer + Policy

Example

Matrix Size: n

UNCONTROLLED

Block Size: s

```
uint64_t matrix_multiply(const uint64_t * M1, const uint64_t * M2, uint64_t
* M3, int n, int s) {
    s = __spec_check_get_general_i32("s", s);
    int en = s * (n/s);

    for (int kk = 0; kk < en; kk += s) {
        for (int jj = 0; jj < en; jj += s) {
            for (int i = 0; i < n; i++) {
                for (int jjj = 0; jjj < s; jjj++) {
                    int j = jj + jjj;
                    element_t sum = M3[i * n + j];
                    for (int kkk = 0; kkk < s; kkk++) {
                        int k = kk + kkk;
                        sum += M1[i * n + k] * M2[k * n + j];
                    }
                    M3[i * n + j] = sum;
                }
            }
        }
    }

  return 0;
}
```

Memory access pattern decided by **Matrix Size** and **Block Size**

Specialization Points

Parameters or variables in performance critical code that can be changed or made constant during execution to improve performance

Provide API to developers to indicate specialization points in their codebase

Only compile and optimize the code that is affected by the specialization points

Observation 1: Optimal Config changes with workload + deployment

Deployment \ Workload	n=1024	n=256	n=64
Machine A	32	32	32
Machine B	16	16	4
Machine C	16	16	4
Machine D	32	4	4
Machine E p-core	32	4	2
Machine E e-core	64	4	4

n = 64

Observation 2: Optimal performance requires compile-time changes

Deployment	Runtime : Compiler Cost Ratio
Machine A	1.61x
Machine B	2.62x
Machine C	2.64x
Machine D	3.48x
Machine E p-core	3.36x
Machine E e-core	2.69x

Converting config parameters into compile time constants requires fewer cycles to do 1 multiplication as compared to leaving them as runtime parameters

Specialization Runtime

Inputs: Code with specialization points

Hooks control runtime and allow points to be selected for specialization

- Analyzer tracks the state of all the specialization points in the input code
- Specializer modifies the handler to specialize selected specialization points
- LLVM JIT recompiles the modified handler code with compiler optimizations

Output: Specialized and optimized code

Explorer

Iteratively explores different specialization for the various specialization points

Selects the best performing specialization and finalizes the handler code to use it

Policy

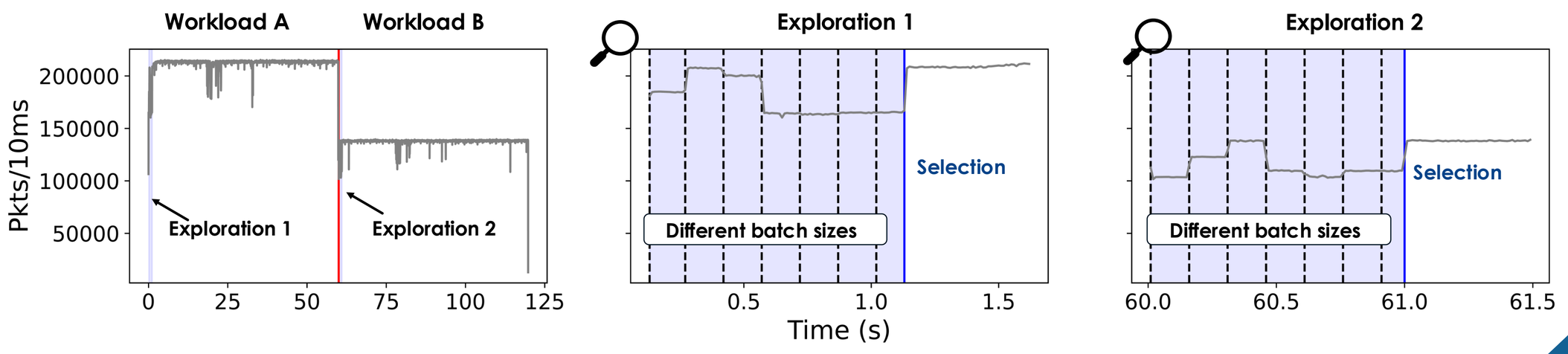
Defines the specialization space to explore

Defines the selection criteria (the end-to-end measurement to optimize)

Provides exploration params

With Iridescent

Network Address Translator



With Iridescent