Re-Activate Harmony

Philip Yang
Jeff Hollingsworth
phi@cs.umd.edu
University of Maryland, College Park

Outline

- Active Harmony review
- Software design
- Experiment result
- Future work
- Q&A

Motivation

- Program can be transformed by changing
  - Template parameters, user flags
  - Loop unrolling, tiling
  - Threads distribution, SIMD
- Transformation interactions are complex
  - **Unrolling** can affect **tiling** by change memory access pattern
  - **Tiling** may limit **ILP** (Instruction Level Parallelism)
- Static analysis alone can't capture all these
- Exhaustive search is too expensive

Our Approach

- Parallel Rank Ordering (PRO)
  - Unconstraint optimization
  - Tunes the program while it is running
  - Utilizes parallelism for parallel program
  - Gives good initial performance
  - Quickly approaches optimal configuration
- The core of Active Harmony

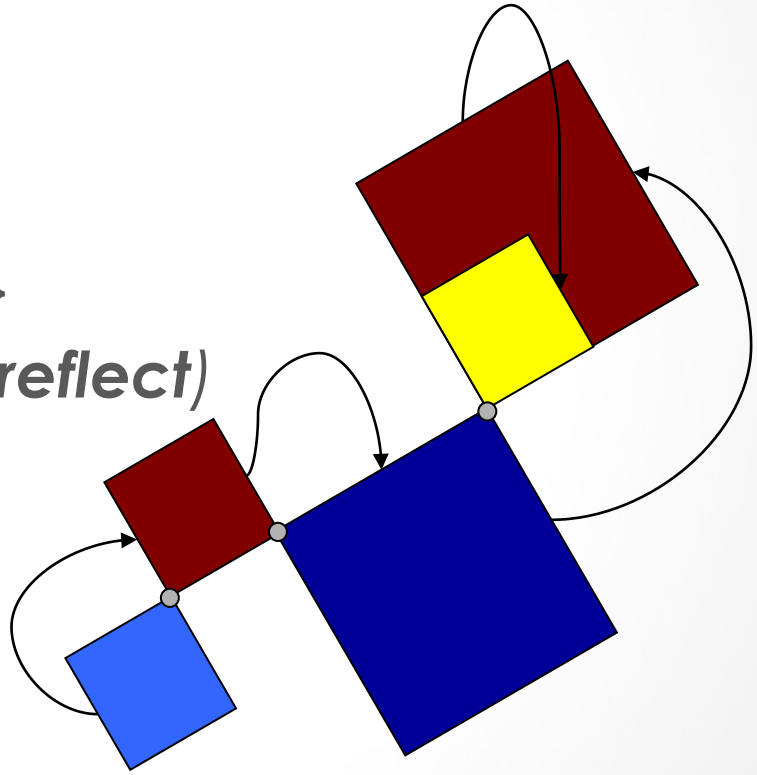
PRO Search: Input

- X Simplex of kN vertices
  - vertices in N-dimensional parameter space
  - kN = #processor
- f_t Performance Measurement Function
  - Smaller value reflects better performance
  - Time, error, etc.
- P Projection to parameter space
  - Force constraints

PRO Search: Simplex

Maintain simplex

1. Measure Proj{***original***}
2. Measure Proj{***reflect***}
3. IF min(***reflect***) < min(***original***)
    1. Measure Proj{***expand***}
    2. IF min(***expand***) < min(***reflect***)
        1. Accept ***expand***
    3. ELSE
        1. Accept ***reflect***
4. ELSE
    1. Accept ***shrink***



PRO Search: Convergence

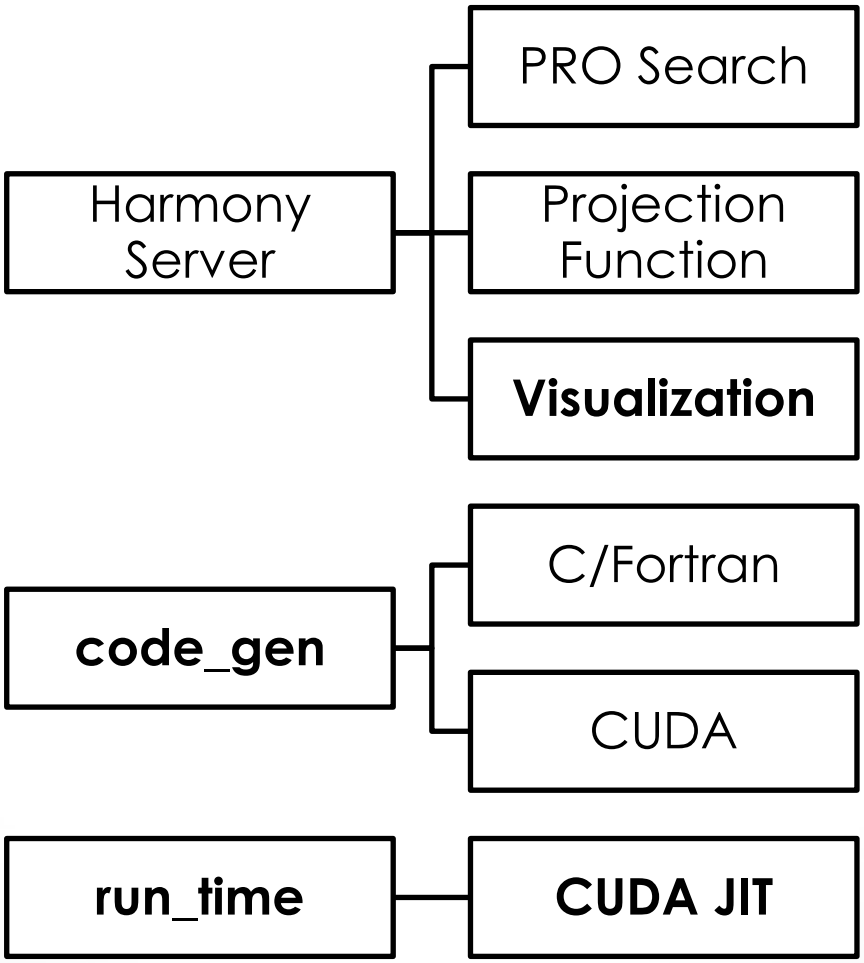
- Termination Criteria
  - Simplex collapsed to a single point
  - We reached max number of iterations
- Sensitive to initial simplex
- Empirically, terminates pretty quickly
- If f_t is continuously differentiable,

$$\lim_{k \to \infty} \inf \left\| \nabla f_t(X_k) \right\| = 0$$

System Design Goals

- Portability
  - Core components in C++ and Python
  - Minimize dependencies
- Loosely coupled modules
  - Change the search algorithm without affecting the rest
- Dynamically load CUDA code via JIT
  - Code server compiles .cu to .ptx, assembly file for CUDA
  - Use CUDA Driver API's JIT engine
- Visualization

System Architecture



code_gen

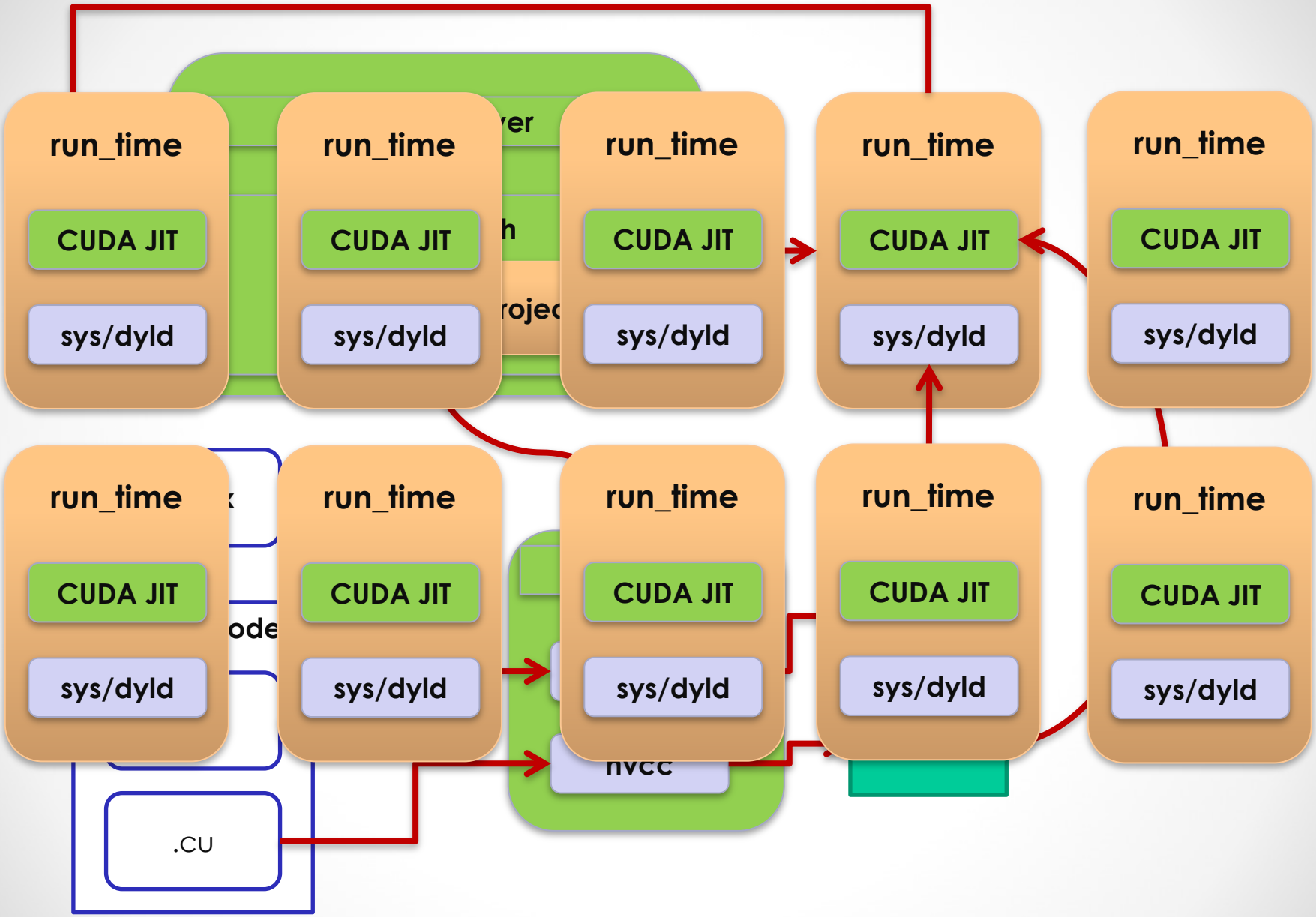
- Many transformations require recompilation
- Recompile the code to dynamic libraries and load them while running
- For CUDA
  - Use nvcc to compile .cu to .ptx
  - Load .ptx at runtime with JIT provided in the CUDA Driver API

code_gen: CUDA JIT

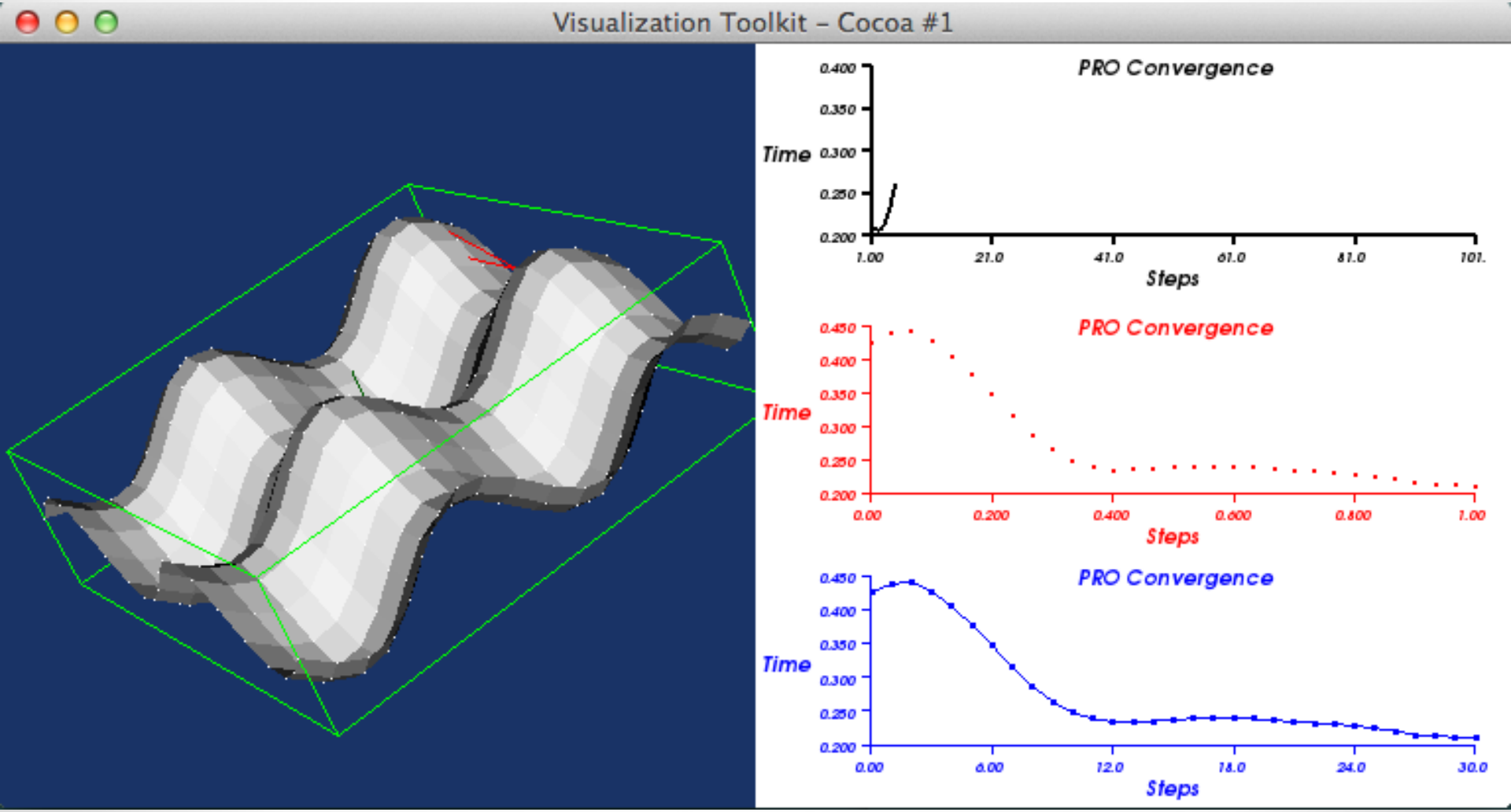
```
#include "code_gen.h"

int main(int argc, char** argv)
{
   CUdrvBase driver;
   driver.init(argc, argv); // initialize device
   driver.load("simple_kernel.ptx"); // load ptx code

   // Initialize data …
   // launch kernel
   cuLaunchGrid(driver.getKernel(), num_blocks, num_threads);
```

code_gen: CUDA JIT

```
>        max #threads per block:   512
>        max block dimension:      512 x 512 x 64
>        max grid  dimension:      65535 x 65535 x 1
>        shared memory per block:  16384
>        total constant memory:    65536
>        warp size (SIMD width):   32
>        memory pitch:             2147483647
>        registers per block:      16384
>        clock rate:               1.24 GHz
>        texture alignment:        256
> compiling source code matrixMul_kernel.cu into ptx
> nvcc -ptx matrixMul_kernel.cu -o matrixMul_kernel.ptx -DBLOCK_SIZE=16
> compilation finished
> loading ptx code:|matrixMul_kernel.ptx|
> found kernel matrixMul
> kernel matrixMul loaded
> 1 new kernel(s) loaded
CUDA kernel launched
```


run_time
CUDA JIT
sys/dyld
run_time
CUDA JIT
sys/dyld
run_time
CUDA JIT
sys/dyld
run_time
CUDA JIT
sys/dyld
run_time
CUDA JIT
sys/dyld
run_time
CUDA JIT
sys/dyld
run_time
CUDA JIT
sys/dyld
run_time
CUDA JIT
sys/dyld
run_time
CUDA JIT
sys/dyld
nvcc
.CU

Visualization



Experiment

- PRO Convergence Test
  - Randomly initialized Simplex size = 40 * dim(X)
  - Random initialization of simple

Dimension / function	2	3	4	5
Poly deg=10	22	22	22	22
Sin(sum(X))	23	23	22	21
Sin(X^2)	21	24	23	20

Experiment: DGEMM

- GPU: 295 GTX, CPU: 8 Xeon cores
- Parameters:
  - Array padding
  - Activate page-locked memory
  - Splitting Matrix A or B
  - How much to split to GPU

	original	harmonized	Speed up
Performance	89 GFLOPS	145 GFLOPS	1.63x

Future Work

- Improve search algorithm
  - Adaptive step size for PRO
  - Harmonize harmony
- Statistical learning approach
  - Different code might utilize different search strategies
  - Predict the performance
- Performance Energy trade-off
  - Game theory: Nash/Stackelberg Equilibrium

Q & A

Thank You

Phil

Philharmonic

Harmony