Finding and Optimizing Phases in Parallel Programs

Jeffrey K. Hollingsworth <hollings@cs.umd.edu>
Ray Chen <rchen@cs.umd.edu>
Phases of UMD CS


AV Williams: 1987-2018

Iribe Center: 2018-

**CS@UMD: Future is Exciting**

- Largest Major on campus (over 2,800 undergrads, plus 400+ Computer Engineering)
- New Building in 2018
- Hiring $O(10)$ New Faculty in couple of years
- New Big Data Masters & Certificate Programs
Motivation

• HPC programs often contain “phases”
  – Dynamic execution context
  – Each have distinct performance traits

• Particularly problematic if inside a time-step loop
  – Short phases confound tools
  – Difficult to analyze a rapidly changing landscape
  – Worse if phases are nested
while (locDom->time() < locDom->stoptime())
{
    TimeIncrement(*locDom);
    LagrangeLeapFrog(*locDom);
}
Automatic Phase Identification

• My Failed Prior Attempts
  – IPS-2
  – Paradyn’s Performance Consultant

• Solution
  • Automatic identification is hard, rely on experts for annotations
  • Create virtual phases by stitching together

(c. 1990) (c. 1995)
while (locDom->time() < locDom->stoptime()) {
    cali::Annotation region1("tuner.communication").begin();
    TimeIncrement(*locDom);
    region1.end();

    cali::Annotation region2("tuner.computation").begin();
    LagrangeLeapFrog(*locDom);
    region2.end()
}

Guided Phase Identification
Performance Landscape

Contextual Timeline

Actual Timeline

Contextual Timeline

2.5KB Per Iteration

3,700KB Per Iteration
Cross-Domain Analysis

• Utilize experts during development
  – Library writers specify tuning variables
  – Application writers specify code regions

  – Phase dictates different performance context
    • Even though the same function is being called

My application has three phases

I know what variables affect FFTW performance

I know what variables affect MPI performance

I know what variables affect BLAS performance
Integration Work

• Special annotation types identify:
  – Tunable variables
  – Code regions that should enable tuning

• New Caliper tuning service
  – Listens for and reacts to special annotations
  – Calls Active Harmony to perform search
3D Fast Fourier Transform

- FFT in 3 dimensions
  - Composed of three 1 dimensional FFT’s
  - Data is redistributed among processes between FFT’s

FFTz  A2A1 (blocking)  FFTy  A2A2 (blocking)  FFTx
Computation/Communication Overlap

FFTz

A2A1 (blocking)

FFTy

A2A2 (blocking)

FFTz

A2A1 (non-blocking)

FFTy1

FFTy2

A2A2 (non-blocking)

FFTx

FFTz

A2A1

FFTy

A2A2

FFTx
Auto-tuning Opportunities

- FFTz
- A2A1 (non-blocking)
- FFTy1
- FFTy2
- A2A2 (non-blocking)
- FFTx
- FFTz & Pack
- Unpack & FFTy1
Online Auto-Tuning
Phase Aware Tuning

• Improvements over offline (non-phase) tuning
  – Reduce search dimensions from 24 to 16
  – 40% fewer search steps needed to converge
  – Equivalent performance after convergence
• Eliminates need for training runs
  – Don’t allocate thousands of nodes to train
Offline Auto-Tuning Cost

![Graph showing cumulative fraction of tuning executions over the number of tuning steps for different performance scenarios: converge, best, 5% worse, 10% worse, 20% worse, 50% worse.](image)
Conclusion

• Phases are key for HPC analysis tools
  – Rely on human guidance through annotations
  – Virtualizing repeated phases helps many types of tools

• Annotations unite cross-domain expertise
  – Libraries annotate variables to analyze
  – Application annotate regions to analyze

• Currently analyzing other HPC codes