Adaptive Tuning of Parallel Programs with CnC

Concurrent Collections (CnC) 2016

Murali Emani
Computing Landscape is Dynamic

- Parallel systems are continuously evolving

- Diverse architectures, workloads and data
Computing Landscape is Dynamic

- Parallel systems are continuously evolving
- Diverse architectures, workloads and data
- Execution environment is dynamic!
- Need adaptive parallelism partitioning and mapping.
Adaptive Parallelism Mapping

- Co-executing programs
- Varying degree of resource contention

- Compute/memory/IO bound
- Phased behavior

- Recurring upgrades
- Versions compatibility

- Large number of components
- Increased chances of failure

- Varying amount of I/O
- Scalability issues
Adaptive Parallelism Mapping

Program performance is sensitive to the environment

- Co-executing programs
- Varying degree of resource contention

- Compute/memory/IO bound
- Phased behavior

- Recurring upgrades
- Versions compatibility

- Large number of components
- Increased chances of failure

- Varying amount of I/O
- Scalability issues
Adaptive Parallelism Mapping

Tune program for better performance

- Co-executing programs
- Varying degree of resource contention

- Compute/memory/IO bound
- Phased behavior

- Large number of components
- Increased chances of failure

- Recurring upgrades
- Versions compatibility

- Varying amount of I/O
- Scalability issues
Tuners

- Is it enough to tune once and leave as-is?
- Execution environment might change
- Tuning has to adapt to the environment through the program execution
Tuners in CnC

- Several knobs available to tune

- Step priorities
  - Rank steps based on amount / importance of work
    CNC_USE_PRIORITY

- Thread affinity
  - Improve locality and minimize thread movement
    FIFO_AFFINITY / step_tuner::affinity()
    CNC_PIN_THREADS
Tuners in CnC

- Number of threads
  
  ```
  CnC::debug::set_num_threads
  CNC_NUM_THREADS
  ```

- Tag-Ranges / Tuning Ranges
  - Group of steps instead of single instances

- Partitioning ranges

- Hierarchical CnC?
Current work

(1) **Optimal Tuner Selection**

Determining
— which parameters to tune and
— how much to tune
- Each tunable parameter may have different impact

- Strong / Weak Correlation analysis

- For example,
  - (a) #threads (b) affinity and (c) partitioner
  - Fibonacci series, with different combinations gave up to 2x speedup !!
Machine learning could be of help


- **ISPASS’12**: “Using Utility Prediction Models to Dynamically Choose Program Thread Counts.”, Moore, R. W. and Childers, B. R


- **PLDI’14**: “Adaptive, Efficient, Parallel Execution of Parallel Programs”, Sridharan, G. Gupta, and G. S. Sohi.

- **PLDI’15**: “Celebrating Diversity: A Mixture of Experts Approach for Runtime Mapping in Dynamic Environments”, M. Emani and M. O’Boyle

- **LCPC ’16**: “Mapping Medley: Adaptive Parallelism Mapping with Varying Optimization Goals”, M. Emani
$N$ tuning knobs

$T_1$

$T_2$

$T_n$
ML model

$T_1$

$T_2$

$T_n$

$N$ tuning knobs
ML model

Code features (step/item/tag collections)

System features

$T_1$

$T_2$

$T_n$

$N$ tuning knobs
ML model

Code features
(step/item/tag collections)

System features

$T_1$

$T_2$

$T_n$

N tuning knobs
ML model

Code features (step/item/tag collections)

System features

$T_1$

$T_2$

$T_n$

$N$ tuning knobs
(2) How to determine optimal #threads
Step collection

Schedule all instances

Schedule range of instances
How it works

Configuration generator → Run → Monitor

Hints → Tuner
Configuration generator

Run

Monitor

Tuner

- May take time to reach optimal configuration
- Machine Learning can help reach quicker
Approach – Machine Learning

- Hand crafted solutions infeasible

- Train offline, deploy online
Challenges

- **Training Data**
  - Experiment Design space
  - Prune data

- **Supervised / Unsupervised**
  - Semi-supervised ?
  - Start with a learned heuristic,
  - update and re-learn as and when required
- Active / Passive invoke
  - *Heartbeat* mode – monitor at regular intervals, or
  - As and when required

- Ensure no additional overhead
Summary

- Tuning needs to be adaptive
- Faster decision to tune can deeply impact performance
- Machine Learning could be one way forward.

Murali Emani,
Lawrence Livermore National Laboratory,
emani1@llnl.gov