Finding Chapel's Peak

Using Auto-tuning to Optimize Chapel Programs

Ray S. Chen <rchen@cs.umd.edu>
Jeff Hollingsworth, UMD
Michael P. Ferguson, LTS
Larry Michele, LTS
Motivation

• Portability is a primary goal for Chapel
  o Chapel is architecture agnostic
    o CRAY, Intel, NVIDIA, multi-core, many-core, accelerator, etc.

• Program optimization often isn’t portable
  o Hardware specific issues often at the heart
    o Cache line saturation
    o Data cache size chunking

• Is it possible to achieve both of these goals?
  o Portable source code optimized for local hardware
long *data = malloc( input_size );
int max_idx = input_size / sizeof(long);

for( steps = 0; steps < SOME_BIG_NUMBER; ++steps )
  ++data[ (++i * 16) % max_idx ];
The Auto-tuning Solution

• Active Harmony is an auto-tuning framework
  o Parameter space defined by orthogonal tunable variables
  o Each variable requires a range of valid values
• Augmenting Chapel
  o Proposed syntax changes:
    ```
    config const someArg = 5 \textbf{in} 1..100 \textbf{by} 2;
    ```
• Any Chapel program now an auto-tuning target
  o Program executed iteratively in search of optimal values
  o Active Harmony provides Tuna for this purpose
• If range was added in the Chapel source code:

```bash
> tuna --chapel ./quicksort
```

![Graph showing runtime vs. threshold with the best runtime at 16 threads, 241 tasks, and threshold of 16 with a runtime of 1.78s.]
Climbing Higher

• Data parallel loops (forall)
  o Parallelism based on global variables
    o dataParTasksPerLocale
    o numThreadsPerLocale
  o What if multiple forall loops compete?
    o What is the optimal task distribution strategy?
    o What about nested forall loops?

• Can auto-tuning solve this problem?
  o Investigations to be conducted in the coming year

• See you at SC13!
THANK YOU