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

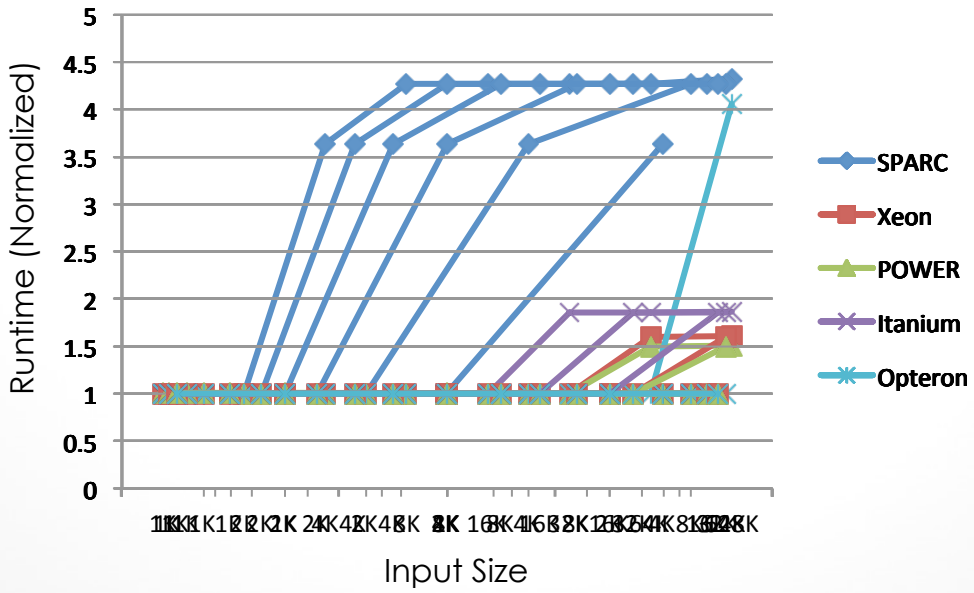
Motivation

- Portability is a primary goal for Chapel
  - Chapel is architecture agnostic
    - CRAY, Intel, NVIDIA, multi-core, many-core, accelerator, etc.
- Program optimization often isn't portable
  - Hardware specific issues often at the heart
    - Cache line saturation
    - Data cache size chunking
- Is it possible to achieve both of these goals?
  - Portable source code optimized for local hardware

Data Cache Optimization

```
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

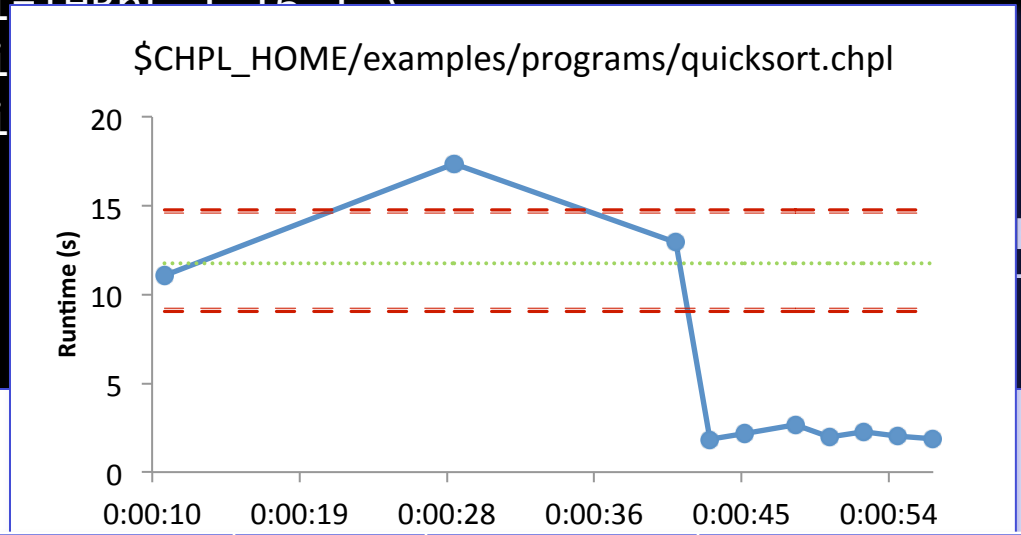
- Active Harmony is an auto-tuning framework
  - Parameter space defined by orthogonal tunable variables
  - Each variable requires a range of valid values
- Augmenting Chapel
  - Proposed syntax changes:

```
config const someArg = 5 in 1..100 by 2;
```

- Any Chapel program now an auto-tuning target
  - Program executed iteratively in search of optimal values
  - Active Harmony provides Tuna for this purpose

Tuning Quicksort

```
> tuna -i-THRpL 1,16,1 \
       -i                                    RpL \
       -i                                    TSKpL \
```



Threads	Tasks	Threshold	Best Runtime
16	241	16	1.78s

- If range was added in the Chapel source code:

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

Climbing Higher

- Data parallel loops (forall)
  - Parallelism based on global variables
    - dataParTasksPerLocale
    - numThreadsPerLocale
  - What if multiple forall loops compete?
    - What is the optimal task distribution strategy?
    - What about nested forall loops?
- Can auto-tuning solve this problem?
  - Investigations to be conducted in the coming year
- See you at SC13!

THANK YOU