Towards the Co-evolution of Auto-tuning and Parallel Languages

Introduction

There exists a gap between the massive parallelism available from today’s HPC systems and our ability to efficiently develop applications that utilize such parallelism. Emerging parallel programming languages such as Chapel aim to narrow this gap through high-level abstractions of data and control parallelism. However, optimizing such programs requires the compiler to make decisions based on information unavailable at compile time. Consider thread and chunk size, as seen in the figures below.

By evolving the Active Harmony auto-tuning framework (detailed on the left) along with the Chapel parallel programming language (detailed on the right), such decisions could be deferred until run time when it’s possible to test for optimal values.

What is the optimal thread count and data chunk size for a given application?

Performance of single-node LULESH [1] written in Chapel with respect to lightweight thread count (tasks) and data chunk size (granularity)

Can auto-tuning be used to mitigate these performance issues?

Using Active Harmony to search the task granulaty parameter space

Conclusion

These experiments demonstrate the utility of auto-tuner and parallel language co-evolution. Looking ahead, the next step in the evolution involves auto-tuner detection of tunable variables within a program, if defined.

We believe wider support for auto-tuning constructs within parallel languages will enable HPC programmers to productively focus on issues of logic rather than performance. Moreover, tighter integration with auto-tuning can enable performance gains within a single execution [2], bringing parallel languages that much closer to closing the performance gap.