Exploring Programming Languages on LLNL Platforms

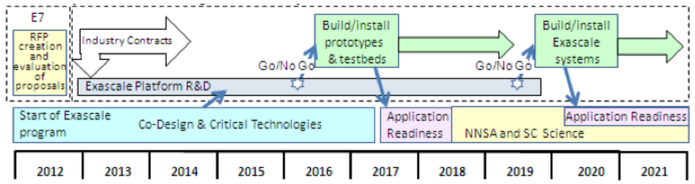
Stephanie Labasan†

Evi Dube, Tom Spelce❖, Kim Yates*

Institute for Scientific Computing Research†, University of the Pacific†, Livermore Computing❖, Center for *Applied Scientific Computing**

Research the Chapel language and explore its performance on LLNL high performance computing platforms in various configurations. Research was completed by writing basic programs in the language and then working with the HPC Challenge benchmarks to compare the programming model with alternative strategies.

Future of Computers: Exascale



- Characteristics of future computer systems:
  - multi-core processors with hundreds of cores/chip
  - performance driven by parallelism
  - subject to frequent faults and failures
- System size and node architecture expected to change dramatically
- Multiple memory types
- Worsening of Memory:FLOPS ratio
- Challenges expected to be within the new node rather than across nodes
- Two options to utilize exascale system:
  - Current practice of MPI
  - Unified programming models at global level

DGEMM Results

- Experiment Settings
  - Environmental Setup: Cab, UC Berkeley's GASNet communication library, Chapel v1.4, Chaos 4 x86-64 bit
  - Chapel Compiler: The flag "—fast" turns off a number of runtime checks are enabled by default for safety, including checks for out-of-bounds array accesses, null pointer dereferences, and violations of locality assertions
- Number of Nodes = Number of Locales
- Single precision matrix multiplication (Chapel does not yet have double precision capabilities)

Matrix Multiplication Calculations

Version 1

```
for j in 1..n {
  for k in 1..n {
    for l in 1..n {
      C(j,k)+=A(j,l)*B(l,k);
    }
  }
}
```

Version 2

```
for j in 1..n {
  for k in 1..n {
    var sum: real=0;
    for l in 1..n {
      sum+=A(j,l)*B(l,k);
    }
    C(j,k)=sum;
  }
}
```

Time (s) for Matrix Multiplication Execution

# Locales	Version 1	Version 2
1	16.9928	11.1264
2	17.8697	11.9178
4	16.3857	12.387
8	17.4369	11.9566
16	16.4915	11.6284

Why Chapel?

- Cascade High-Productivity Language
- Four productivity goals:
  - Programmability - Portability
  - Performance - Robustness
- Provides a global view for expressing data structures and control flow of the program
- Operates under a multiresolution philosophy permitting users to initially write very abstract code and incrementally add more detail
- Parallel features most directly influenced by ZPL, HPF, and Cray MTA™/Cray XMT™ extensions to C and Fortran

Discussion

- Varying amounts of locales does not affect time
- Issue with caching when variable, sum, is not declared – faster time because value of sum is most likely stored in registers
- Cost associated with executing in multilocale directly from the command line vs. msub

Future Work

- Continue writing Chapel in HPCC benchmarks, such as HPL and PTRANS
- Explore meaning of uninode vs. multilocale
- Investigate purpose of using multiple locales
- Look at timings for larger numbers of locales
- Monitor as PGAS alternatives to the present programming model using MPI

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-POST-569613