Strongly Performing Python Implementation of the HPC Challenge

Interactive Supercomputing, Inc.

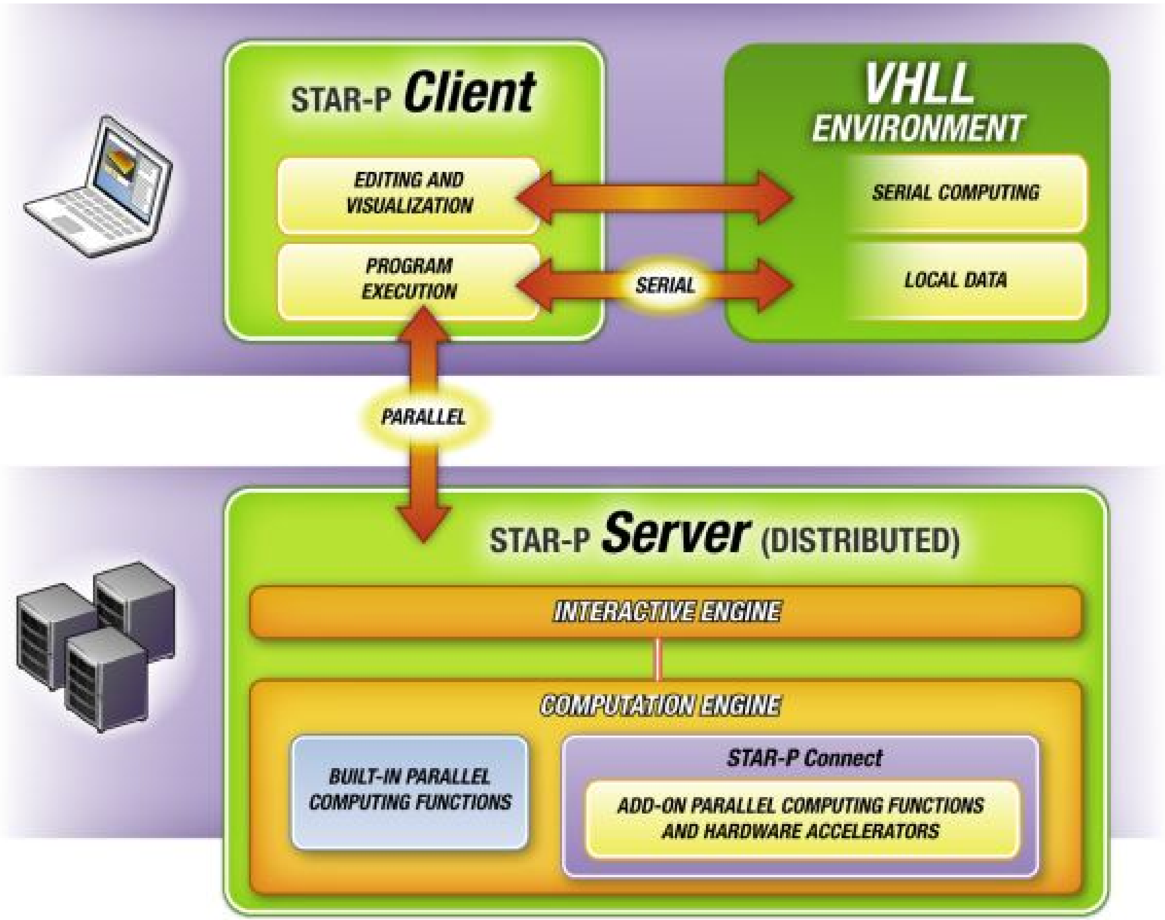
Star-P enables Python, MATLAB®, and R users to go parallel easily with competitive performance.

Approach to HPC Challenge

- Create Python version of 4 HPCC benchmarks
  - HPL, Stream, Random Access, and FFT
  - Why Python?
- Parallelize with Star-P constructs
- Measure and tune

Star-P Basics:

Bridges the gap between desktop tools and parallel computing systems



Value proposition

- Rapid, interactive apps development
- Potent high-level parallel abstractions
- Minimize code changes
- High speed and/or large memory
- MATLAB® and Python clients today, R soon
- Scales to 100s of cores, >4TB memory
- Extensible with existing serial or MPI-parallel libraries

INTERACTIVE supercomputing

Star-P/Python Parallel Constructs

Task-Parallel

- Iterations clearly separable
- Use Star-P's parallel iterator

Data Parallel

- Large monolithic data
- Create distributed arrays
  - Distributed attribute propagates to result variables

HPL Source Code

```
def run_hpl(n, nr, tol=16):
  """
  Run the High-performance LINPACK test on a matrix of size n x n, nr
  number of times and ensures that the the maximum of the three
  residuals is strictly less than the prescribed tolerance (defaults
  to 16).

  This function returns the performance in GFlops/Sec.
  """
  a = random.rand(n, n);
  b = random.rand(n, 1);
  x,t = iterate_func(nr, linalg.solve, a, b)

  r = dot(a, x)-b
  r0 = linalg.norm(r, inf)
  r1 = r0/(eps * linalg.norm(a, 1) * n)
  r2 = r0/(eps * linalg.norm(a, inf) * linalg.norm(x, inf) * n)

  performance  = (1e-9 * (2.0/3.0 * n * n * n + 3.0/2.0 * n * n) *
   nr/t)
  verified     = numpy.max((r0, r1, r2)) < 16

  if not verified:
    raise RuntimeError, "Solution did not meet the prescribed tolerance
   %d"%tol
  return performance
```

STREAM Source Code

```
def run_epstream(n, nr):
  """
  Run the embarrasingly parallel stream benchmark on vectors of size
  n, nr number of times.

  This function returns the performance of the benchmark in
  GFlops/second.
  """
  s = random.rand(1);
  a = random.rand(n);
  b = random.rand(n);
  c,t = iterate_func(nr, lambda s, a, b: s*a+b, s, a, b)

  performance = (1e-9) * 24.0 * nr * n / t

  return performance
```

Random Access Source Code

```
import time, optparse
import starp as sp

def update_state(ran, idx, table_size):
    sp.runCommand('rng_update_state', ran, idx, table_size);

def update(table_size, n_in, n_out):
    t1 = 0;
    t0 = time.time()
    t = sp.arange(table_size)
    t1 += (time.time() - t0)
    ran = sp.zeros(n_in, )
    idx = sp.zeros(n_in, )

    for outer in xrange(n_out):
        update_state(ran, idx, table_size)
        t0 = time.time();
        t[idx] ^= ran
        t1 += (time.time() - t0);

    return 1.0e-9 * n_in * n_out/float(t1);

def run_random_access(n, nr):
    n_in = 1024;
    n_out = nr/n_in;

    if n_out * n_in != nr:
        raise ValueError("Number of updates must be evenly divisible by %d" %
   n_in)

    return update(n, n_in, n_out);
```

FFTE Source Code

```
def run_fft(n, nr, tol=16):
  """
  Run the one-dimensional FFT benchmark on a vector of size n, nr
  number of times and verifies that the inverse transforms recreates
  the original vector upto a tolerance, tol (defaults to 16).

  This function returns the performance in GFlops/sec.
  """
  a = random.rand(n,1)
  b, t = iterate_func(nr, fft.fft, a)

  log2n = math.log(n)/math.log(2)
  performance = 1e-9 * 5.0 * n * log2n * nr/t
  verified    = linalg.norm(a - (fft.ifft(b))) / (eps * log2n) < tol

  if not verified:
    raise RuntimeError, "Solution did not meet the tolerance %d"%tol

  return performance
```

Product Scalability:
Does this work in any other industry?

How the coffee industry treats someone ordering a Large

How the computing industry treats VHLL language users wanting a Large

Code Attributes

Benchmark	*SLOC (Python/ Star-P)*	*SLOC (MPI)*	*Distance to Desktop*
framework	63	?	2
HPL	13	15608	0
STREAM	6	658	0
RandomAccess	46 (+71 C++)	1883	6 (+71 C++)
FFTE	8	1747	0

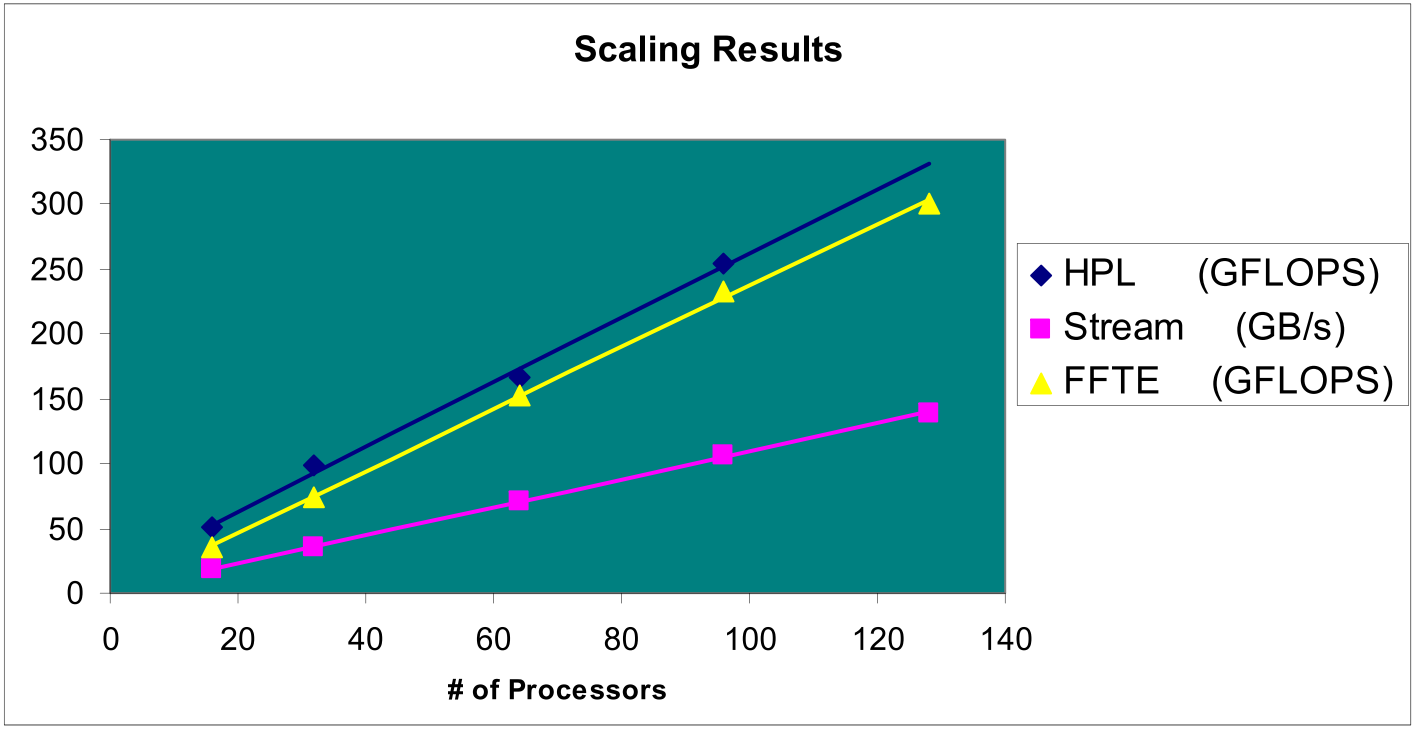
- Implemented with Star-P 2.5.1 (currently shipping product)
- Developed on small in-house system, scaled directly to 128-core system at SDSC
- Difference from desktop, in framework for HPL/Stream/FFTE

```
if nproc == 0:
    from numpy import *
else:
    from starp import *
```

  - Can exert greater control with more code changes
- RandomAccess:
  - Not a good match for current Python/Star-P
  - Used custom 27-line C++ kernel

Performance

#cores	HPL (GFLOPS)	Stream (GB/s)	FFTE (GFLOPS)	RandomAccess (GUPS)
16	50.7	17.894	35.627	0.00118
32	98.901	35.626	74.037	0.00194
64	165.969	70.769	152.574	
96	254.221	106.504	232.475	
128		139.475	299.976	



- Strong absolute performance
- Strong scalability

Relevance for General HPC

- HPCC benchmarks (except RandomAccess) lend themselves to trivial task- or data-parallel expression
  - Data analysis codes are similar, and need rapid development
- Typical HPC apps have more complex data sharing patterns and depend more on many simpler functions, not one large function

Star-P enables Python, MATLAB®, and R users to go parallel easily with competitive performance.

For our full description, go to
www.InteractiveSupercomputing.com/applibrary/