Recursion and slicing impacts on performance in Chapel, C, and Fortran

Nelson Luís Dias[1]

[1]Department of Environmental Engineering, Federal University of Paraná, Brazil

: nelsonluisdias@gmail.com : www.nldias.github.io

ChapelCon '25, Octorber 10 2025

Context

Array slicing is a nice feature in modern languages

In

- Python,
- R,
- Fortran,
- and Chapel

it allows for elegant and concise expression of array operations.

However, the performance of slicing in Chapel lags behind some other languages (known issue)

See for example
https://stackoverflow.com/questions/48243717/array-slicing-performance-in-chapel
and
https://chapel.discourse.group/t/new-issue-improve-the-performance-of-slices-and-rank-change-operations/30503

Reason: slicing in Chapel generates an array view.

However, the performance of slicing in Chapel lags behind some other languages (known issue)

See for example
https://stackoverflow.com/questions/48243717/array-slicing-performance-in-chapel
and
https://chapel.discourse.group/t/new-issue-improve-the-performance-of-slices-and-rank-change-operations/30503

Reason: slicing in Chapel generates an array view.

This presentation is not intended as criticism of Chapel.

Instead, it calls attention to a current limitation in Chapel and shows alternatives to obtain good performance using an artificial and simple example.

Implementation

Steps

Repeat 10 times: for each language: Chapel, C (gcc), C (clang) and Fortran (gfortran),

1. Fill an array with 64,000,000 random 64-bit floating point numbers.
2. Sum the array sequentially.
3. Sum the array recursively, passing indices.
4. Sum the array recursively, with slicing (only Chapel and Fortran).

Arrays are filled with the same "Even Quicker Generator" in Press et al. (1992, p. 284). Since my version of Fortran does not have unsigned 32-bit integers, the Fortran test programs are linked with the object file from the C random generator source, via

```
gcc -O3 -c qdran.c
```

Computer specs

```
  System: Kernel: 6.14.0-33-generic arch: x86_64 bits: 64
  compiler: gcc v: 13.3.0 clocksource: tsc

  Desktop: Cinnamon v: 6.4.8 tk: GTK v: 3.24.41 wm:
  Muffin v: 6.4.1 vt:7 dm: LightDM v: 1.30.0
  Distro: Linux Mint 22.2 Zara base: Ubuntu 24.04 noble

Machine:
  Type: Desktop Mobo: Gigabyte model: B660M GAMING X

CPU:
  Info: 16-core (8-mt/8-st) model: 12th Gen Intel Core i9-12900K bits

Memory: total: 64 GiB note: est. available: 62.57 GiB used: 3.94 GiB (6.3%)
```

Code snippets: non-recursive, sequential

```
// ---------------------------------------------
// --> seqsum: non-recursive sequential sum
// ---------------------------------------------
proc seqsum(
   const ref a: [] real
   ): real where a.rank == 1 {
   var sum: real = 0.0;
   for x in a do {
      sum += x ;
   }
   return sum;
}
```

Code snippets: recursive, index and slicing

```
// ------------------------------------------
// --> recsum: recursive sum using indices
// ------------------------------------------
proc recsum(
   const ref A: [] real,
   const in low: int,
   const in high: int
   ): real where A.rank == 1 {
   if low == high then {
      return A[low];
   }
   var mid = (low+high)/2;
   return recsum(A,low,mid) +
      recsum(A,mid+1,high);
}
```

```
// ------------------------------------------
// --> precsum: recursive sum using slices
// ------------------------------------------
proc precsum(
   const ref A: [] real
   ): real where A.rank == 1 {
   var low = A.dim(0).low;
   var high = A.dim(0).high;
   if low == high then {
      return A[low];
   }
   var mid = (low+high)/2;
   return precsum(A[low..mid]) +
      precsum(A[mid+1..high]);
}
```

Results

Running times

All

Lang	algorithm	sum	elapsed time
Chapel	sequential	3.2002e+07	0.19449900 s
C gcc	sequential	3.2002e+07	0.32388200 s
Clang	sequential	3.2002e+07	0.31900100 s
Fortran	sequential	0.3200E+08	0.32615100 s
Chapel	recursive/index	3.2002e+07	1.30998300 s
C gcc	recursive/index	3.2002e+07	0.90463100 s
Clang	recursive/index	3.2002e+07	1.22071800 s
Fortran	recursive/index	0.3200E+08	0.76957000 s
Chapel	recursive/slices	3.2002e+07	34.87501100 s
Fortran	recursive/slices	0.3200E+08	0.77340900 s

Best

Algorithm	Language	elapsed time
overalll	Chapel	0.19449900 s
sequential	Chapel	0.19449900 s
recursive/index	Fortran	0.76957000 s
recursive/slice	Fortran	0.77340900 s

Chapel & Clang recursive/index are close: coincidence or result of the same LLVM backend?

Conclusions

Conclusions

- In this example, Chapel is the best overall for the straightforward solution of summing an array.
- Fortran's performance is almost the same between recursive sum passing indices and recursive sum with slicing.
- Chapel's performance with slicing is much poorer than Fortran's. But note that this is an artificial case with huge amounts of slicing involved.

Thanks for the attention.

References

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C; The Art of Scientific Computing* (2nd ed.). Cambridge University Press.