ChapelPerf: A Performance Suite for Chapel

Ricardo Jesus and Michèle Weiland

EPCC, The University of Edinburgh

CHIUW 2022
Outline

• Introduction
• Benchmark description
• Porting experience
• Preliminary performance results
• Conclusions and future work
• Application porting and performance optimisation are hard, especially in HPC
  • Plethora of system configurations, variety of codes they are meant to run, etc
• High-level parallel programming languages, libraries, and runtimes, like Chapel, help at coping with this complexity. But
  • How do we know if a given parallel programming framework is good for a specific class of problems/codes?
  • How do we know if it delivers good performance across different system architectures?
  • How do we determine how it compares to other alternatives?

The answer, unsurprisingly, is via benchmarks.
RAJAPerf: The RAJA Performance Suite

- RAJAPerf\(^1\) is a benchmark suite originally developed for RAJA
- It consists of **over 50 loop-based kernels** extracted from HPC applications, other benchmark suites, and similar sources
- Each kernel is **implemented in a number of “variants”** corresponding to different programming models/frameworks

<table>
<thead>
<tr>
<th>Base_SEQ</th>
<th>Lambda_SEQ</th>
<th>RAJA_SEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base_OpenMP</td>
<td>Lambda_OpenMP</td>
<td>RAJA_OpenMP</td>
</tr>
<tr>
<td>Base_OMPTarget</td>
<td>Lambda_OMPTarget</td>
<td>RAJA_OMPTarget</td>
</tr>
<tr>
<td>Base_CUDA</td>
<td>Lambda_CUDA</td>
<td>RAJA_CUDA</td>
</tr>
<tr>
<td>Base_HIP</td>
<td>Lambda_HIP</td>
<td>RAJA_HIP</td>
</tr>
</tbody>
</table>

- The loop body of each kernel is implemented similarly across variants
- A checksum is computed per kernel variant to ensure its correct execution

\(^1\)https://github.com/LLNL/RAJAPerf
ChapelPerf: “The Chapel incarnation of RAJAPerf”

• https://github.com/rj-jesus/chapelperf

• A mostly complete port of RAJAPerf (v0.11.0) to Chapel

• Developed mainly to evaluate the performance of Chapel compared to other parallel programming models and across different system architectures

• We have fully implemented two variants of each kernel: Base_Chpl and Forall_Chpl
  • Working towards implementing more “idiomatic” variants such as Promotion_Chpl and Reduction_Chpl

• Command-line options and outputs are the same as RAJAPerf’s to simplify its usage
Porting experience

- Overall a straightforward process
- Most kernels easily ported by adapting the C++ code to Chapel
- Similar experience porting the logic around the kernels (i.e. the way RAJAPerf is structured, how the execution is driven, and so on)
  - Mostly a matter of mapping C++ features such as inheritance, polymorphism, and various containers, to Chapel’s analogues

Example with Apps_FIR kernel

```chapel
for(RepIndex_type irep = 0; irep < run_reps; ++irep) {
  //pragma omp parallel for
  for(Index_type i = ibegin; i < iend; ++i ) {
    Real_type sum = 0.0;
    for(Index_type j = 0; j < coefflen; ++j )
      sum += coeff[j]*in[i+j];
    out[i] = sum;
  }
}
```

```chapel
for 0..<run_reps {
  for i in ibegin..<iend {
    var sum: Real_type = 0.0;
    for j in 0..<coefflen do
      sum += coeff[j]*in_[i+j];
    out_[i] = sum;
  }
}
```

```chapel
for 0..<run_reps do
  forall i in ibegin..<iend do
    out_[i] = + reduce (coeff*in_[i..]);
```
Porting challenges: Aliasing array views

• Some RAJAPerf kernels create “aliasing views” over a common array
  • Trivial in C/C++ since we can declare arbitrary pointers to an array directly. Example from MASS3DPA¹ to the right

```
double sm1[MDQ * MDQ * MDQ];
double(*DDQ)[MD1][MQ1] = (double(*)(MD1)[MQ1])sm1;
double(*QQQ)[MQ1][MQ1] = (double(*)(MQ1)[MQ1])sm1;
double(*QDD)[MD1][MD1] = (double(*)(MD1)[MD1])sm1;
```

• Implementing something similar in Chapel in a straightforward manner does not seem to be possible currently. We have found two main workarounds:
  • Utilising inline procedures to capture the underlying array and encapsulate the necessary index arithmetic (right)
  • Using a wrapper class where the index arithmetic is encapsulated in the class’s this method²

```
var sm1: [0..<MDQ*MDQ*MDQ] real;
inline proc DDQ(i,j,k) ref return sm1[(i*MD1+j)*MQ1+k];
inline proc QQQ(i,j,k) ref return sm1[(i*MQ1+j)*MQ1+k];
inline proc QDD(i,j,k) ref return sm1[(i*MD1+j)*MD1+k];
```

It would be good if arrays in Chapel supported this type of aliasing natively

¹https://github.com/LLNL/RAJAPerf/blob/v0.11.0/src/apps/MASS3DPA.hpp#L190
²https://gitter.im/chapel-lang/chapel?at=6196745fabdd6644e390f5b9
Porting challenges: long double

- RAJAPerf uses long double’s extensively to compute checksums of runs
- Chapel does not support such a type neither natively nor as a “C type”
- But, relatively easy to work around
  - The implementation of long double in LCALS\(^2\) is mostly complete
  - We extended it to increase interoperability with other Chapel types and to enable long double’s to be used for input/output

```chapel
operator :(s: string, type t: longdouble) {
    var ld: longdouble;
    assert(sscanf(s.localize().c_str(), "%Lf", c_ptrTo(ld)) == 1);
    return ld;
}
```

```chapel
proc longdouble.writeThis(f) throws {
    var buf = new c_array(c_char, 255);
    var ret = snprintf(buf:c_ptr(c_char), buf.size:size_t, "%Lf", this);
    writer <> buf:c_string: string;
    return ret;
}
```

\(^2\)https://github.com/chapel-lang/chapel/blob/1.25.0/test/release/examples/benchmarks/lcals/LongDouble.chpl
Preliminary performance results (I)

- Comparison between sequential and parallel variants
- Chapel 1.26 using GCC for backend

<table>
<thead>
<tr>
<th>System</th>
<th>Processor</th>
<th>Compiler</th>
<th>Opt. flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCHER2 (HPE Cray EX)</td>
<td>EPYC 7742</td>
<td>GCC 11.2.0</td>
<td>-O3 -march=native/--fast</td>
</tr>
<tr>
<td>Fulhame (HPE Apollo 70)</td>
<td>ThunderX2</td>
<td>GCC 10.1.0</td>
<td>-O3 -mcpu=native/--fast</td>
</tr>
<tr>
<td>Isambard 2 (HPE Apollo 80)</td>
<td>A64FX</td>
<td>GCC 10.3.0</td>
<td>-O3 -mcpu=native/--fast</td>
</tr>
</tbody>
</table>

- Most kernels do well compared to the reference C++ sequential/OpenMP versions
- **A few kernels do far worse**
  - Some kernels can run 1000x slower than the reference C++ versions
  - Slowdowns are more common when comparing the parallel variants and more pronounced on kernels that belong to the Apps group
- The Arm-based systems tend to do comparatively worse than the x86 one

(plots on next slide)
Preliminary performance results (II)

![Graph showing performance results for different kernels on various architectures]

- **ARCHER2 (HPE Cray EX)**
- **Fulhame (HPE Apollo 70)**
- **Isambard 2 (HPE Apollo 80)**

**Kernels and Applications**

- Algorithm SORT
- Algorithm SORTPAIRS
- Apps DEL DOT VEC 2D
- Apps DIFFUSION3DPA
- Apps ENERGY
- Apps FIR
- Apps HALOEXCHANGE
- Apps HALOEXCHANGE FUSED
- Apps LTIMES
- Apps LTIMES NOVIEW
- Apps MASS3DPA
- Apps PRESSURE
- Apps VOL3D
- Basic DAXPY
- Basic IF QUAD
- Basic INIT3
- Basic INIT VIEW1D
- Basic INIT VIEW1D OFFSET
- Basic MAT MAT SHARED
- Basic MULADDSUB
- Basic NESTED INIT
- Basic PI ATOMIC
- Basic PI REDUCE
- Basic REDUCE3 INT
- Basic TRAP INT
- Lcals DIFF PREDICT
- Lcals EOS
- Lcals FIRST DIFF
- Lcals FIRST MIN
- Lcals FIRST SUM
- Lcals GEN LIN RECUR
- Lcals HYDRO 1D
- Lcals HYDRO 2D
- Lcals INT PREDICT
- Lcals PLANCKIAN
- Lcals TRIDIAG ELIM
- Polybench 2MM
- Polybench 3MM
- Polybench ADI
- Polybench ATAX
- Polybench FDTD 2D
- Polybench FLOYD WARSHALL
- Polybench GEMM
- Polybench GEMVER
- Polybench GESUMMV
- Polybench HEAT 3D
- Polybench JACOBI 1D
- Polybench JACOBI 2D
- Polybench MVT
- Stream ADD
- Stream COPY
- Stream DOT
- Stream MUL
- Stream TRIAD

**Contact**

rjj@ed.ac.uk

**ChapelPerf: A Performance Suite for Chapel**

CHIUW’22
Conclusions and future work

- ChapelPerf is an implementation of the RAJAPerf kernels in Chapel
  - Drop-in replacement for RAJAPerf
  - Enables the comparison of Chapel with many other programming models and frameworks across different systems architectures
- Preliminary results show that Chapel overall does well compared to reference implementations, with exceptions
  - Slowdowns can reach 1000x
  - Chapel on Arm-based systems tends to do comparatively worse
  - These results already offer a pointer to code patterns that might necessitate more optimisation on Chapel
- Next steps
  - Identify and address the factors leading to the occasional reduced performance in Chapel (we are particularly interested in Arm)
  - Implement more variants (idiomatic, GPU-based, multinode?\(^3\))

\(^3\)RAJAPerf recently added preliminary support for this
Questions?

Contact: rjj@ed.ac.uk