LCALS: Livermore Compiler Analysis Loop Suite

David Iten, Elliot Ronaghan, Cray Inc.

CHIUW 2016

May 27th, 2016

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

LCALS: Outline

- LCALS Overview
- LCALS Serial Performance Comparison
- LCALS Parallel Performance Comparison
- Future work

LCALS: Background

- **LCALS: Livermore Compiler Analysis Loop Suite**
  - Loop kernels designed to measure compiler performance
  - Developed by LLNL
  - https://codesign.llnl.gov/LCALS.php

LCALS Code
Richard D. Hornung
LCALS version 1.0
LLNL-CODE-638939
2013

- **Three loop subsets (30 kernels total)**
  - Subset A: Loops representative of application codes
  - Subset B: Simple, basic loops
  - Subset C: Loops extracted from "Livermore Loops coded in C"

- **Each kernel is run for three sizes (Short, Medium, Long)**

- **Each kernel is implemented in a number of "variants"**
  - RAW (traditional C usage), OpenMP, C++ template-based, etc.

LCALS: Chapel Port

- **Chapel LCALS port consists of**
  - ~2400 lines of framework Chapel code
  - ~2200 lines of kernel Chapel code
- **Implements two kernel variants**
  - RAW+Serial: 30 kernels
  - RAW+Parallel: 11 kernels
    - RAW+Parallel kernels are a modified subset of the RAW serial kernels
    - The OpenMP variant in the reference
- **Performance compared vs. reference versions**
  - Executed on one Cray XC40 compute node
  - 24 Intel Xeon cores per node
  - Compiled with: gcc 5.3.0
  - The following charts show results for the “Long” size

LCALS: Serial Kernel Format

- **Each serial kernel follows the pattern:**

C reference

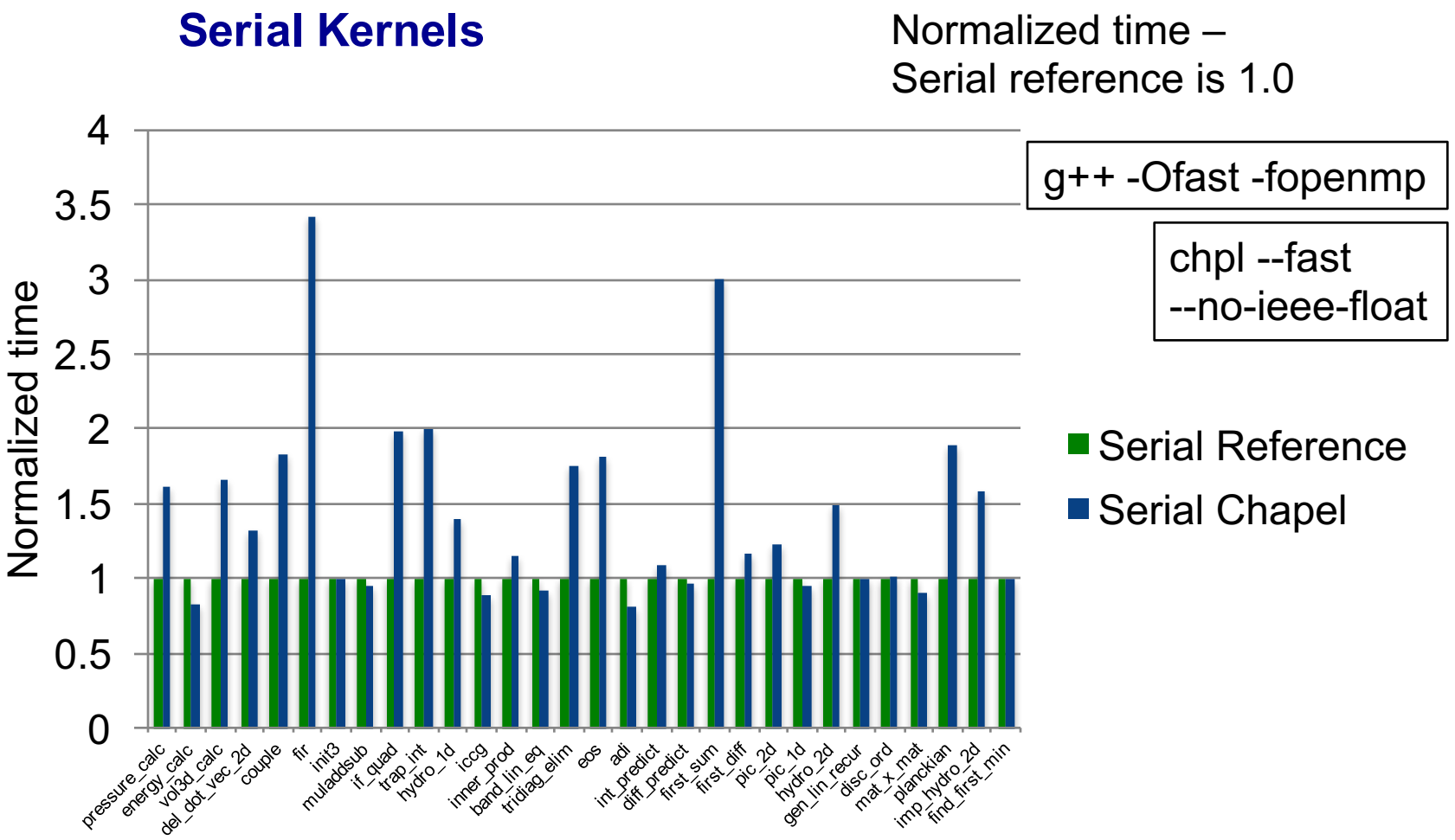
```
initArrays();
startTimer();
for (isamp=0; isamp<num_samples; isamp++) {
  for (i=0; i<len; i++) {
    <main kernel body>
  }
}
stopTimer();
```

Chapel

```
initArrays();
startTimer();
for isamp in 0..#num_samples {
  for i in 0..#len {
    <main kernel body>
  }
}
stopTimer();
```

- **A few kernels have additional inner loops**

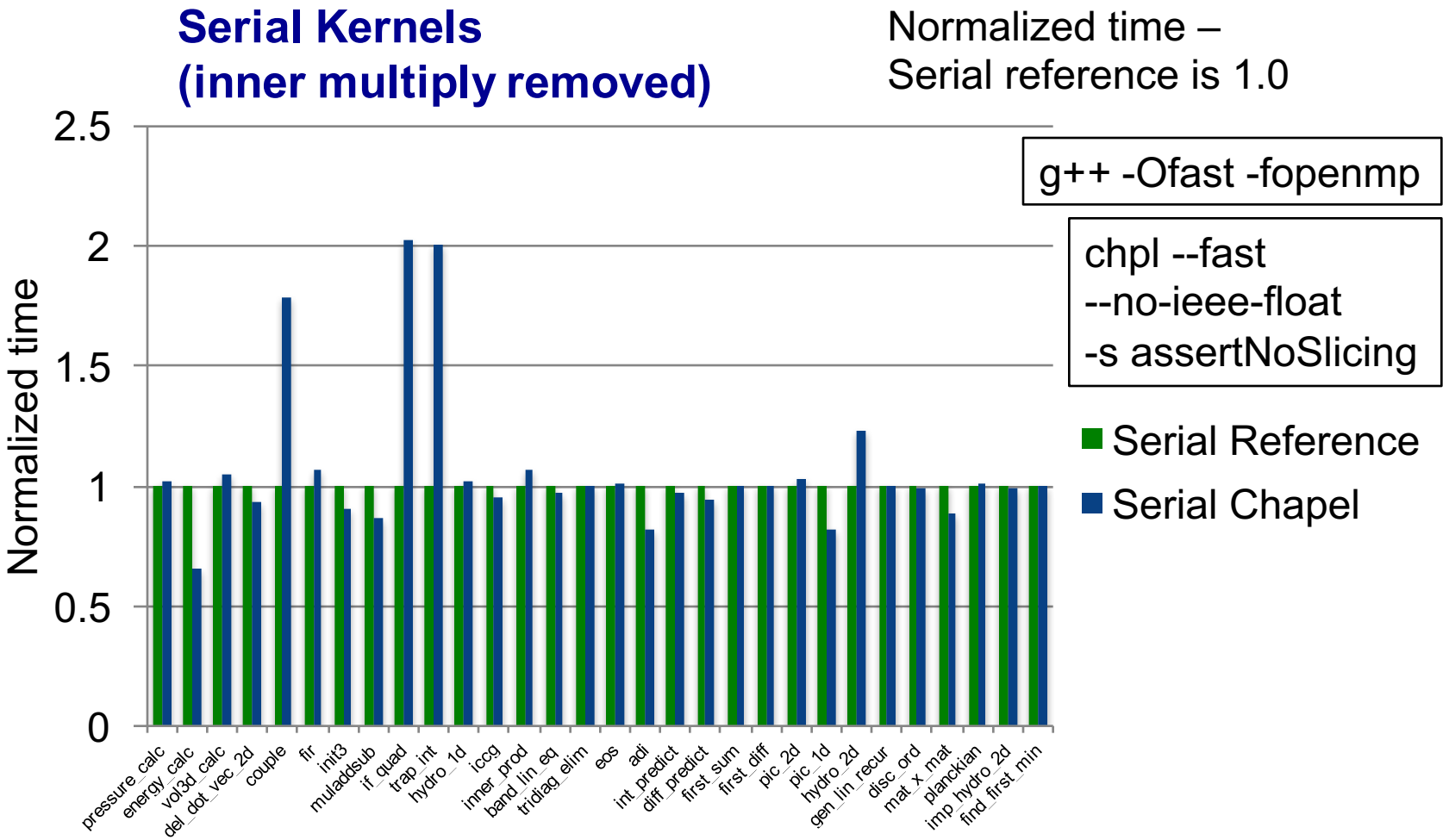
LCALS: Performance Comparison



LCALS: Array Inner Multiplications

- **LCALS serial performance lags C for many kernels**
  - Chapel uses an integer multiply for an array's innermost dimension
    - Unnecessary for typical arrays, only for more advanced ones
    - e.g., rank-change, reindexing of strided slices, …
  - For typical cases, adds overhead relative to C
    - Ongoing work is striving to eliminate multiplies in these cases
  - Meanwhile, can be squashed manually using a config param
    - Results in dramatic serial performance improvements for most loops…
    - …bringing them in line with C except for a few outliers

LCALS: Performance Comparison



LCALS: Parallel Kernel Format

- Parallel kernels are derived from serial kernels
- Usually one parallel loop inside the 'num_samples' loop

C reference

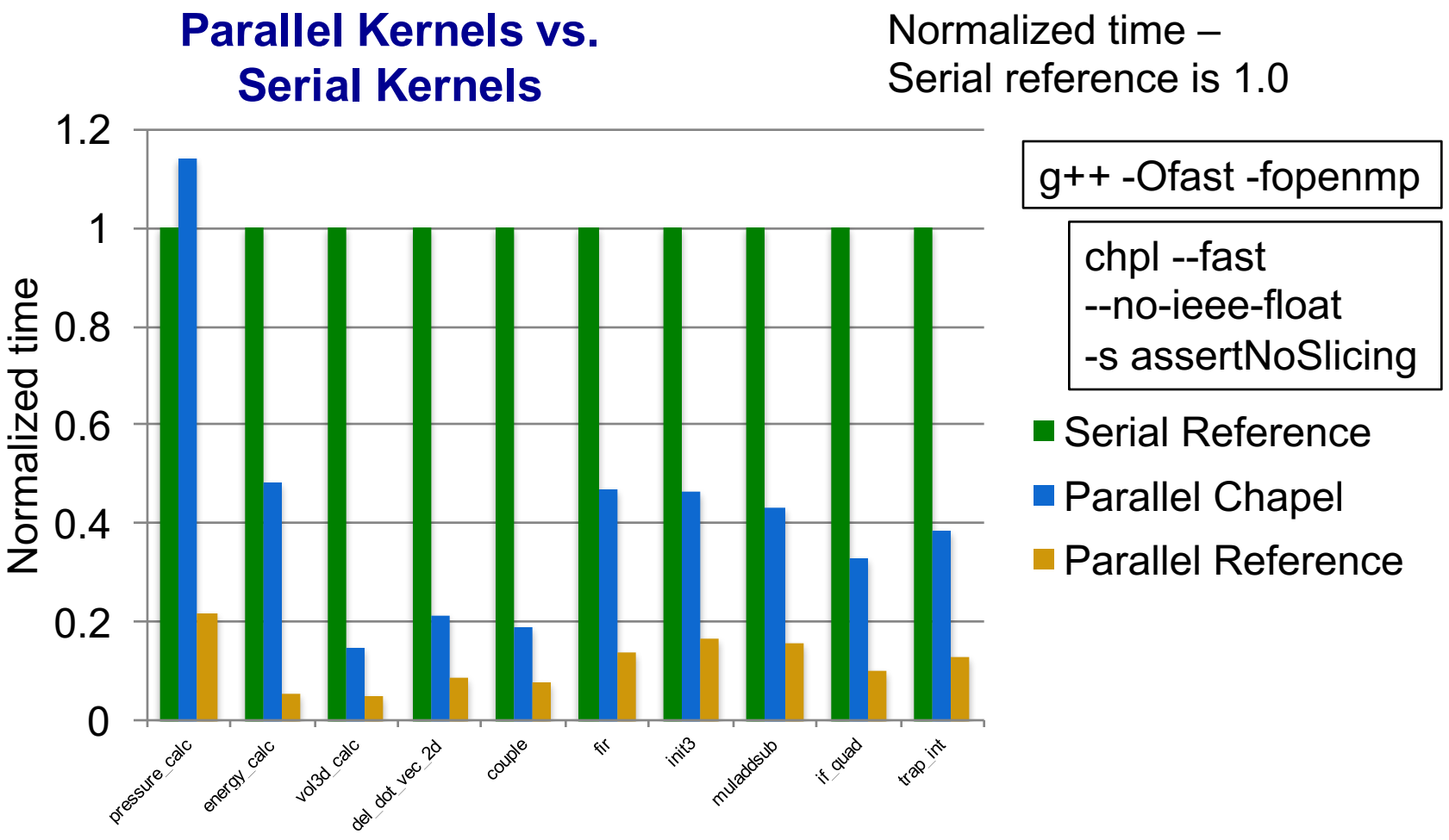
```
initArrays();
startTimer();
for (isamp=0; isamp<num_samples; isamp++) {
  #pragma omp parallel for
  for (i=0; i<len; i++) {
    <main kernel body>
  }
}
stopTimer();
```

Chapel

```
initArrays();
startTimer();
for isamp in 0..#num_samples {
  forall i in 0..#len {
    <main kernel body>
  }
}
stopTimer();
```

- A few kernels have additional inner loops

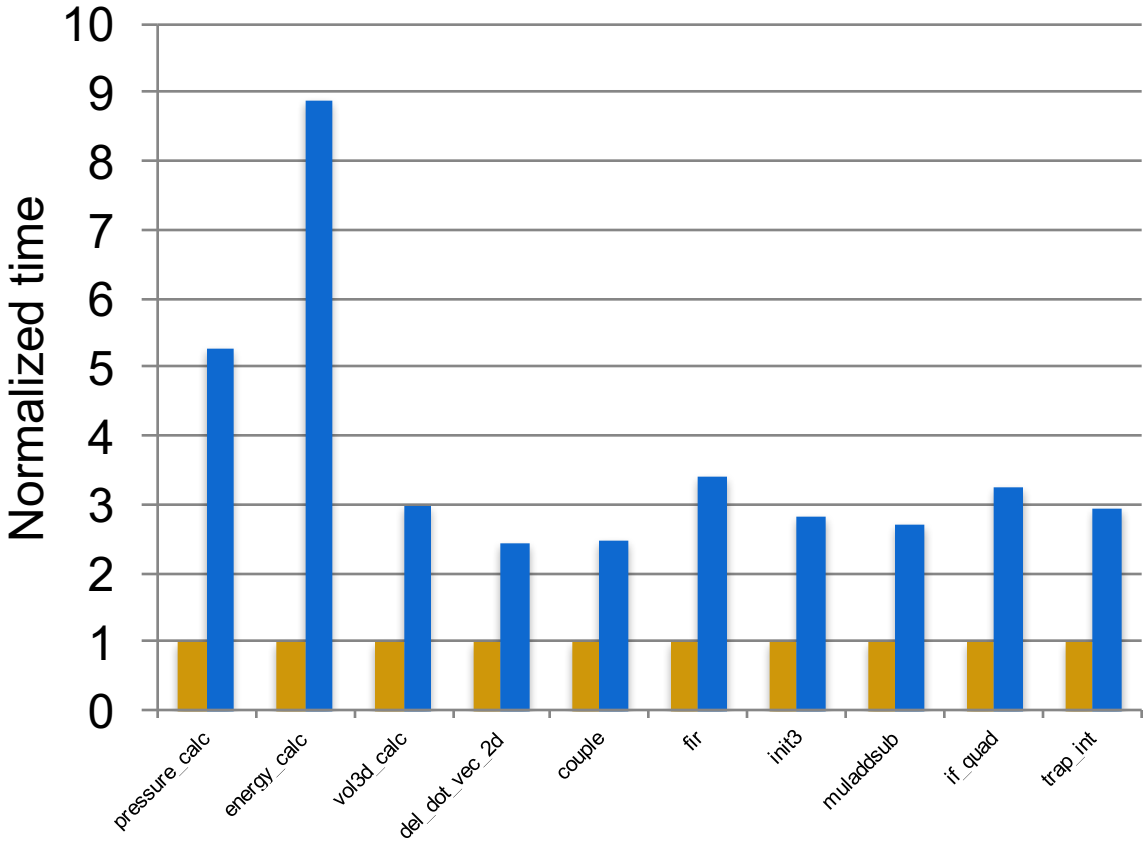
LCALS: Performance Comparison



LCALS: Performance Comparison

Parallel Kernels

Normalized time –
Parallel reference is 1.0



g++ -Ofast -fopenmp

chpl --fast
--no-ieee-float
-s assertNoSlicing

Parallel Reference

Parallel Chapel

LCALS: Array Inner Multiplications

- **Most kernels are ~3x slower than OpenMP reference**
  - …except for pressure_calc (5x) and energy_calc (9x)
  - …the two parallel kernels with multiple inner loops

These used an obvious translation:

OpenMP Reference

```
#pragma omp parallel
{
  #pragma omp for nowait
  for (i=0; i<len; i++) {
    …work1()...
  }
  #pragma omp for nowait
  for (i=0; i<len; i++) {
    ...work2()...
  }
}
```

Parallel Chapel

```
forall i in 0..len {
  …work1()...
}
forall i in 0..len {
  ...work2()...
}
```

Combined foralls

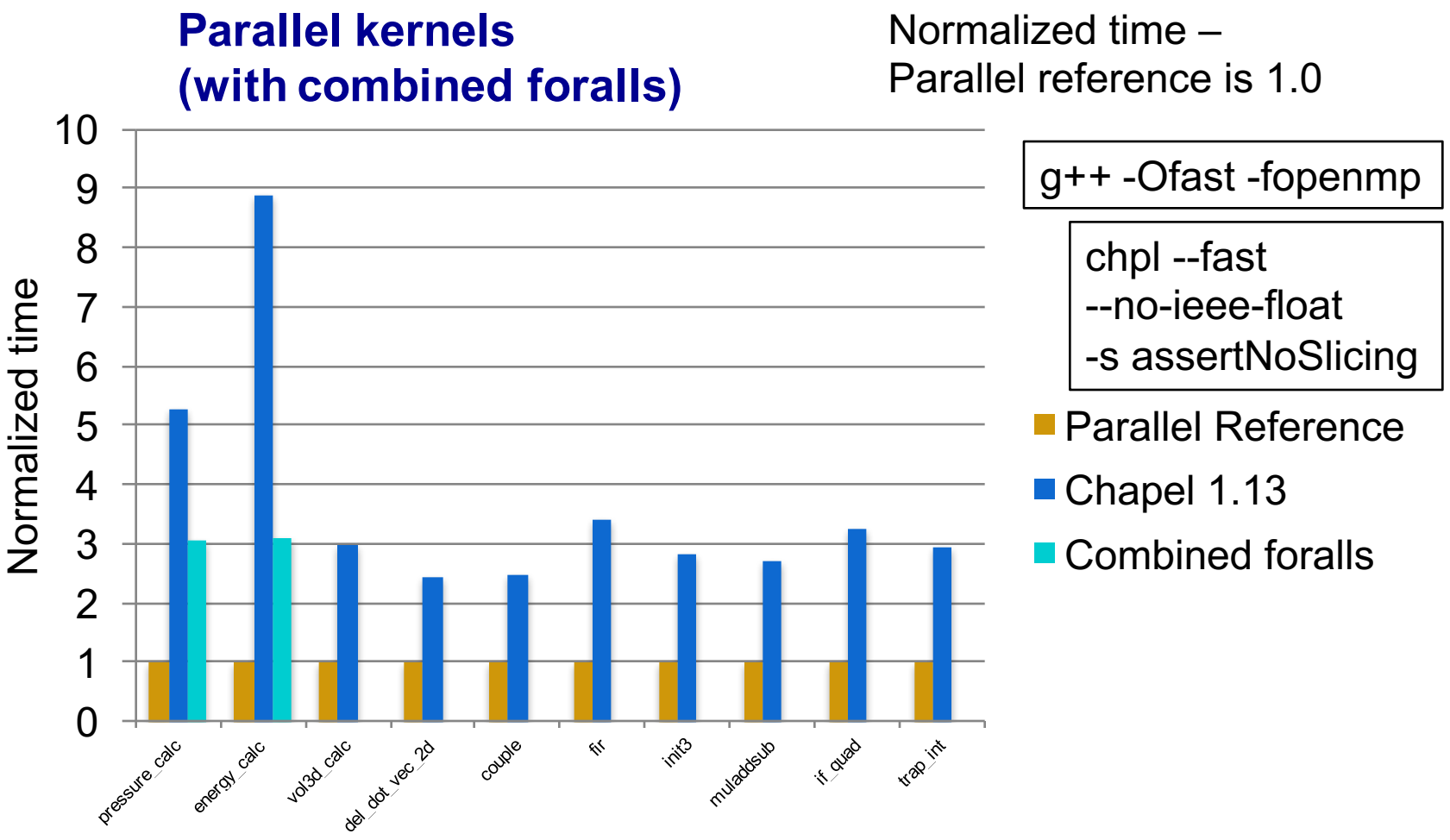
```
forall i in 0..len {
  …work1()...
  ...work2()...
}
```

So instead, combined consecutive forall loops.

nowait clause doesn't have an obvious Chapel equivalent

Not always possible, but works for these kernels.

LCALS: Performance Comparison



LCALS: Parallel loop startup cost

- **'forall' startup is more costly than OpenMP 'parallel for'**
  - Allocates and initializes an argument bundle per task
  - Should be able to eliminate most of this overhead
- **The parallel loops are short, even in the 'long' size**
  - Around 45,000 iterations in each parallel loop
  - … split between 24 tasks
  - … and repeated 15,000 times
  - Magnifies startup cost difference
- **Increasing the iteration count would mask the difference**
  - But need task startup improvements to match OpenMP at small sizes

LCALS: Next Steps

- **Eliminate array inner multiplies when unnecessary**
  - The -sassertNoSlicing hammer is too big
  - A new feature is planned to do this in a more principled way
- **Optimize the last few uncompetitive serial kernels**
- **Improve task startup overhead**
  - If the problem size is increased this overhead is masked
  - But should be able to match OpenMP performance for small loops too
- **Explore more elegant Chapel loop expressions**
  - Use whole-array operations, array slicing, etc.
  - Make sure the elegant versions perform well too

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

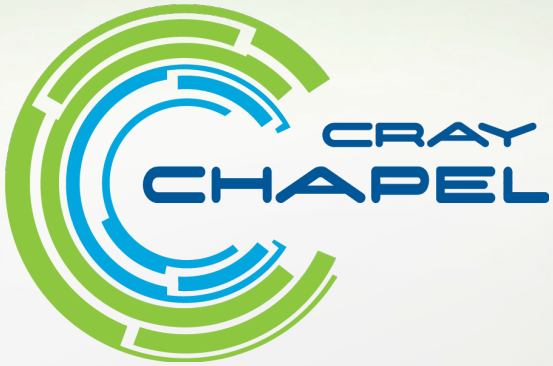
All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.



CRAY
THE SUPERCOMPUTER COMPANY

http://chapel.cray.com chapel_info@cray.com https://github.com/chapel-lang/chapel