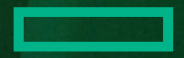

Hewlett Packard
Enterprise

CHAPEL 1.31/1.32 RELEASE NOTES:
COMPILER / RUNTIME / PORTABILITY IMPROVEMENTS

Chapel Team

June 22, 2023 / September 28, 2023

OUTLINE

- Vectorization User Support
- Co-locale Improvements
- ARM Improvements
- Heterogeneous Processing Units
- Other Improvements

VECTORIZATION USER SUPPORT

VECTORIZATION USER SUPPORT

Background and This Effort

Background:

- Many processors support parallelization with vector (SIMD) instructions
  - Compilers can take advantage of this by vectorizing code, and this can improve performance
- It can be difficult to determine if Chapel code has been vectorized

This Effort:

- Added support for an experimental attribute to inspect vector code generation

```
@llvm.assertVectorized      // warns at compile-time if this loop was not vectorized
foreach i in A.domain do
  A[i] = sqrt(i:real(32));
...
foreach i in A.domain {
  @llvm.assertVectorized    // warns at compile-time if this loop was not vectorized
  foreach j in A.domain do
    A[i] += sqrt(j:real(32));
}
```

VECTORIZATION USER SUPPORT

This Effort, Status, and Next Steps

This Effort (continued)**:**

- Added new flags '--llvm-remarks' and '--llvm-remarks-function' to inspect backend optimizations

```
> chpl vector.chpl --fast --llvm-remarks=vector -g
...
vector.chpl:6:0: opt passed for 'loop-vectorize' - vectorized loop
(vectorization width: 4, interleaved count: 2)
...
vector.chpl:14:0: opt missed for 'loop-vectorize' - the cost-model indicates
that vectorization is not beneficial
...
```

- Added preliminary support for '@llvm.metadata' to experiment with code generation

Status: The new features currently only work with the LLVM backend

Next Steps: Stabilize and expand loop attributes

CO-LOCALE IMPROVEMENTS

CO-LOCALE IMPROVEMENTS

Background and This Effort

Background:

- *Co-locales* are locales running on the same node without oversubscription
- Co-locale support was previously limited to Slurm/OFI
- Co-locales were opted into using the CHPL_RT_LOCALES_PER_NODE environment variable
  - Currently limited to one locale per socket

This Effort:

- Extended support to Slurm and PBS launchers on GASNet
- Extended command-line arguments to support specifying co-locales
  - Specifying '-nl NxL' allocates N nodes with L locales each
    - `./myChapelProgram -nl 4x2` *# creates 8 locales on 4 nodes*
    - `./myChapelProgram -numLocales 4x2` *# ditto*

CO-LOCALE IMPROVEMENTS

Impact and Next Steps

Impact:

- Improved ease-of-use by being able to specify co-locales on the command-line
- Improved performance on multi-socket GASNet machines (e.g., dual-socket Xeon 8360Y)

Config	Heap	Stream Throughput
1 locale per node	Fixed	182 GB/s/node
2 locales per node	Fixed	297 GB/s/node
1 locale per node	First-touch	304 GB/s/node
2 locales per node	First-touch	303/GB/s/node

Next Steps:

- Add co-locale support to remaining GASNet launchers
- Support one locale per NUMA domain
  - Modern processors have multiple NUMA domains within a socket
- Explore having Chapel choose a “smart” default number of locales per node via ‘-nl 4x’

ARM IMPROVEMENTS

ARM IMPROVEMENTS

Background and This Effort

Background: In past releases, Chapel had performance issues on ARM systems

- Qthreads tasking layer lacked native context switching for 64-bit ARM, so task creation/switching was slow
  - Especially slow on M1/M2 macs, leading us to use 'fifo' tasking there by default

This Effort: Upgraded to qthreads 1.19, which includes native 64-bit ARM context switching

- Collaborated with qthreads team to validate implementation
- Changed default tasking layer to qthreads on M1/M2 macs

ARM IMPROVEMENTS

Impact: ARM Linux

Impact: Improved qthreads task switching speed on ARM Linux

- Task switching microbenchmark

```
coforall 1..here.maxTaskPar*4 do
  for 1..500_000 do
    currentTask.yieldExecution();
```

Config	w/o fast tasks	w/ fast tasks	improvement
56-core x86 Skylake	4.37s	0.32s	13.6x
64-core ARM ThunderX2	4.85s	0.44s	11.0x
64-core ARM Graviton3	2.67s	0.28s	9.5x
48-core ARM A64FX	11.79s	2.73s	4.3x

ARM IMPROVEMENTS

Impact: ARM Macs

Impact: Significantly improved qthreads task switching speed on ARM Macs

- Task switching microbenchmark

```
coforall 1..here.maxTaskPar*4 do
  for 1..500_000 do
    currentTask.yieldExecution();
```

Config	qt w/o fast tasks	fifo	qt w/ fast tasks	improvement
8P-core ARM M1 Pro	29.63s	8.85s	0.15s	197.5x / 59.0x

ARM IMPROVEMENTS

Impact: Yielding Communications

Impact: Better performance for applications with communication idioms that yield

- Especially those with multiple tasks per core (explicit with oversubscription or implicit from aggregation)
  - e.g., Bale Indexgather on 16-node Cray XC with ARM ThunderX2

Approach	w/out fast tasks	with fast tasks	improvement
ordered	70.7 MB/s/node	84.7 MB/s/node	1.20x
ordered, oversubscribed	86.3 MB/s/node	140.4 MB/s/node	1.63x
unordered	147.5 MB/s/node	152.3 MB/s/node	1.03x
aggregated	1352.0 MB/s/node	1448.5 MB/s/node	1.07x

HETEROGENEOUS PROCESSING UNITS

HETEROGENEOUS PROCESSING UNITS

Background:

- Some processors have processing units (PUs) with different performance profiles
  - e.g., Intel's Alder Lake has 8 cores with 2 performance PUs, and 8 cores with 1 efficiency PU
- This triggered a bug in the runtime while computing the number of inaccessible cores

This Effort:

- Added support for specifying which kind of PU to use via the CHPL_RT_USE_PU_KIND environment variable
  - Must be one of "performance", "efficiency", or "all"
  - Default is "performance"

Impact:

- Allows the user to specify the kind of PUs used by their application

OTHER IMPROVEMENTS

OTHER IMPROVEMENTS

For a more complete list of compiler, runtime, and portability changes and improvements in the 1.31 and 1.32 releases, refer to the following sections in the CHANGES.md file:

- Platform-specific Performance Optimizations / Improvements
- Compilation-Time / Generated Code Improvements
- Generated Executable Flags
- Portability / Platform-specific Improvements
- Compiler Improvements
- Runtime Library Changes
- Launchers
- Developer-oriented changes: Compiler Flags
- Developer-oriented changes: Compiler improvements / changes
- Developer-oriented changes: Runtime improvements

THANK YOU

https://chapel-lang.org
@ChapelLanguage