EVALUATING DATA PARALLELISM IN C++ PROGRAMMING MODELS USING THE PARALLEL RESEARCH KERNELS

Jeff Hammond
Exascale CoDesign, Data Center Group, Intel Corporation

Acknowledgements: Tim Mattson, Rob van der Wijngaart, Alex Duran, Jim Cownie, Alexey Kukanov, Tom Scoglund and the rest of the RAJA team at LLNL, CodePlay SYCL team, ...

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Additional Disclaimer

I am not an official spokesman for Intel. I do not speak for my collaborators, whether they be inside or outside Intel.

I work on system pathfinding and workload analysis, not software products. I am not a developer of Intel software tools.

You may or may not be able to reproduce any performance numbers I report, but the code is on GitHub and I will provide anything else you need to attempt to reproduce my results.

Hanlon's Razor (blame stupidity, not malice).

Title:

Evaluating data parallelism in C++ programming models using the Parallel Research Kernels

Abstract:

Modern C++ provides a wide range of parallel constructs in the language itself, as well as tools to implement general and domain-specific parallel frameworks for both CPUs and accelerators. Examples include Threading Building Blocks (TBB), RAJA, Kokkos, HPX, Thrust, SYCL, and Boost.Compute, which complement the C++17 parallel STL. This talk will describe our attempts to systematically compare these models against lower-level models like OpenMP and OpenCL. One goal is to understand the tradeoffs between performance, programmability and portability in these frameworks to educate HPC programmers. The experiments are based on the Parallel Research Kernels (https://github.com/ParRes/Kernels/), which is a collection of application proxies associated with high-performance scientific computing applications such as partial differential equation solvers, deterministic neutron transport, 3D Fast Fourier Transforms, and dense linear algebra.

Bio:

Jeff Hammond is Senior System Architect in the exascale co-design team at Intel. His research interests include computational chemistry, numerical linear algebra, parallel programming models, and high-performance computing system architecture. He contributes to the development of the open standards for parallel computing (MPI, OpenMP, OpenSHMEM, ISO C++) and a wide range of open-source software (e.g. NWChem). He received the IEEE Technical Committee on Scalable Computing (TCSC) Young Achiever Award for work on massively parallel scientific computing. Prior to joining Intel, he worked at Argonne Leadership Computing Facility. He received his PhD in chemistry from the University of Chicago, where he was a Department of Energy Computational Science Graduate Fellow (DOE-CSGF). For more information, please see https://github.com/jeffhammond.

Outline

- Background and context
- Description of C++ programming models
  - OpenMP™, OpenCL™ (baselines)
  - KOKKOS, RAJA, TBB, parallel STL (primary)
  - SYCL™ (secondary)
- Experimental results with Parallel Research Kernels
  - KNL (Intel® Xeon™ Phi 7250 processor)
  - SKX (Intel® Xeon® Platinum 8180 processor)

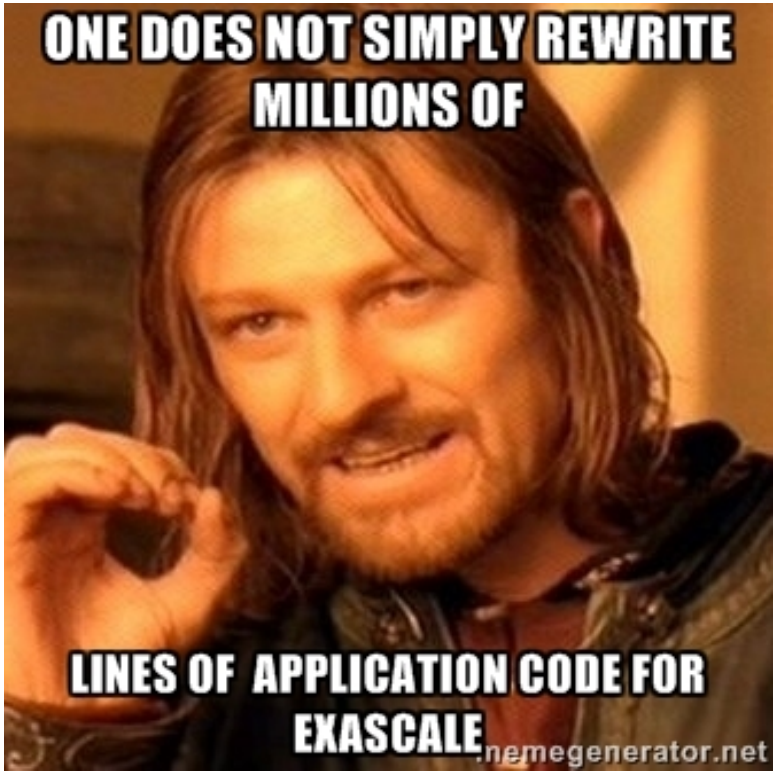
This talk is not intended to be a tutorial!

This talk is not about performance tuning!

HPC software design challenges (2016)

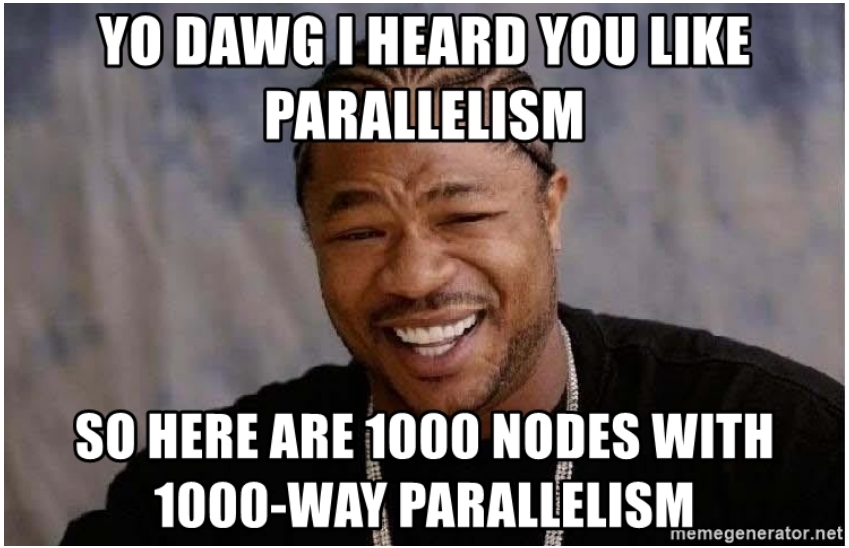
- To MPI or not to MPI…
- One-sided vs. two-sided?
- Does your MPI/PGAS need a +X?
- Static vs. dynamic execution model?
- What synchronization motifs maximize performance across scales?

Application programmers can afford to rewrite/redesign applications zero to one times every 20 years…



HPC software design challenges (2018)

- Intranode parallelism is growing much fast than internode…
- Intranode parallelism is far more diverse than internode parallelism.
  - After ~20 years, internode behavior is converged to some subset of MPI-3.
  - Big Cores, Little Cores, GPU, FPGA all require (very) different programming models.



How do we maximize productivity+performance+portability?

C++ AND PARALLELISM

I study molecular dynamics, but to tell the truth I am interested more in the dynamics than in the molecules, and I care most about questions of principle.

Phil Pechukas, Columbia University Chemical Physics Professor

I study C++ parallelism, but to tell the truth I am interested more in the parallelism than in the C++, and I care most about questions of practice.

Why C++ parallelism?

- C++ is a kitchen sink language – it has pretty much every feature that exists in programming languages (other than simplicity and orthogonality).
- Used across essentially all markets/domains where parallelism or performance matter.
  - Fortran and Rust usage domain-specific.
  - Interpreted languages do not satisfy performance requirements.
- C++ can be extended to do all sorts of things within the language itself. Variadic templates for fun and profit!
- Mattson's Law: No new languages!

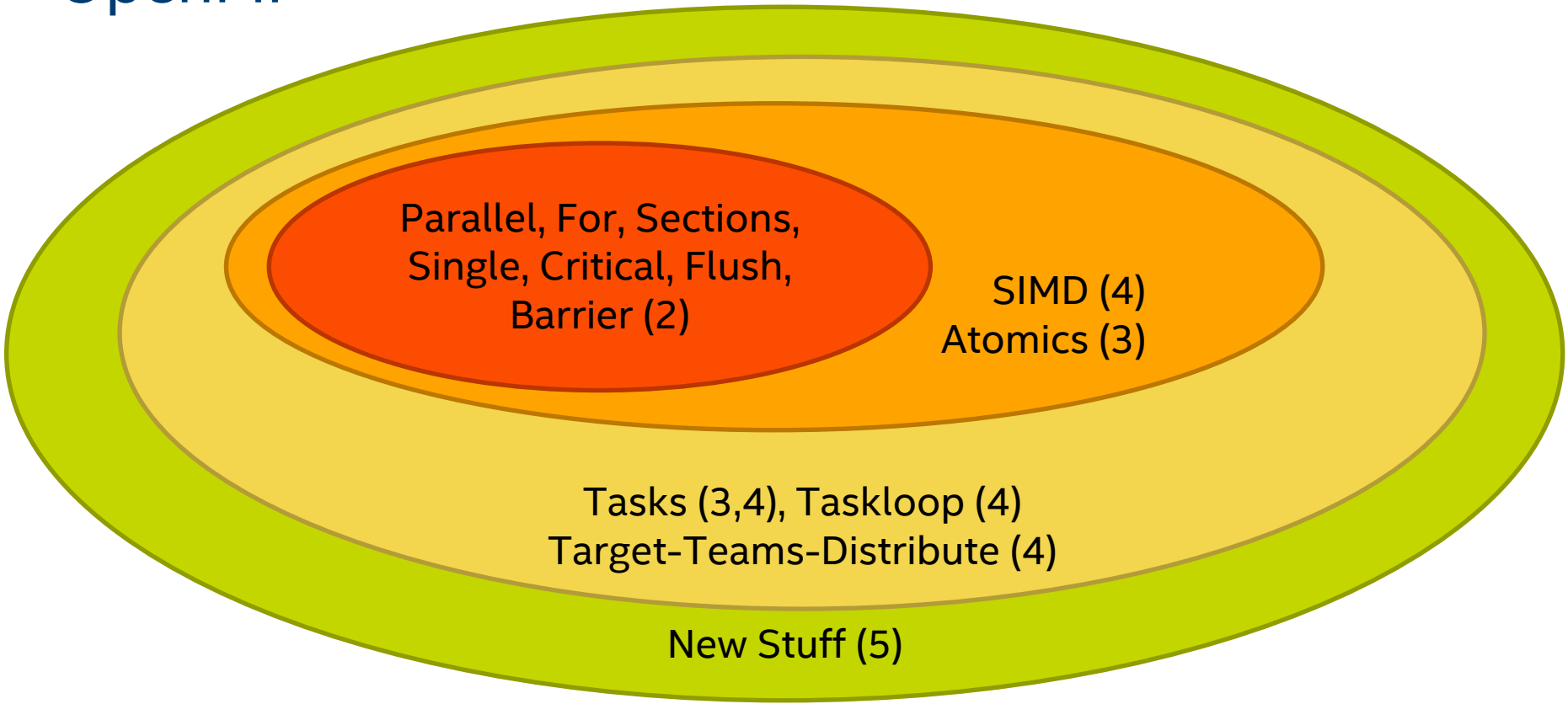
* Named for Tim Mattson. I'm the only one who refers to this as such.

Overview of Parallel C++ models

- TBB (Intel OSS) - parallel threading abstraction for CPU*.
- KOKKOS (Sandia) – parallel execution and data abstraction for CPU and GPU architectures (OpenMP, Pthreads, CUDA, …).
- RAJA (Livermore) – parallel execution for CPU and GPU architectures (OpenMP, TBB, CUDA, …). CHAI adds GPU data abstraction.
- PSTL (ISO standard) – parallel execution abstraction for CPU architectures; designed for future extensions for GPU, etc. (e.g. Thrust and HPX).
- SYCL (Khronos standard) - parallel execution and data abstraction that extends the OpenCL model (supports CPU, GPU, FPGA, …).

PSTL = Parallel STL, meaning C++17 plus executor extensions (proposed for standard)
* We exclude TBB flowgraph support for OpenCL nodes for the purposes of this study.

OpenMP



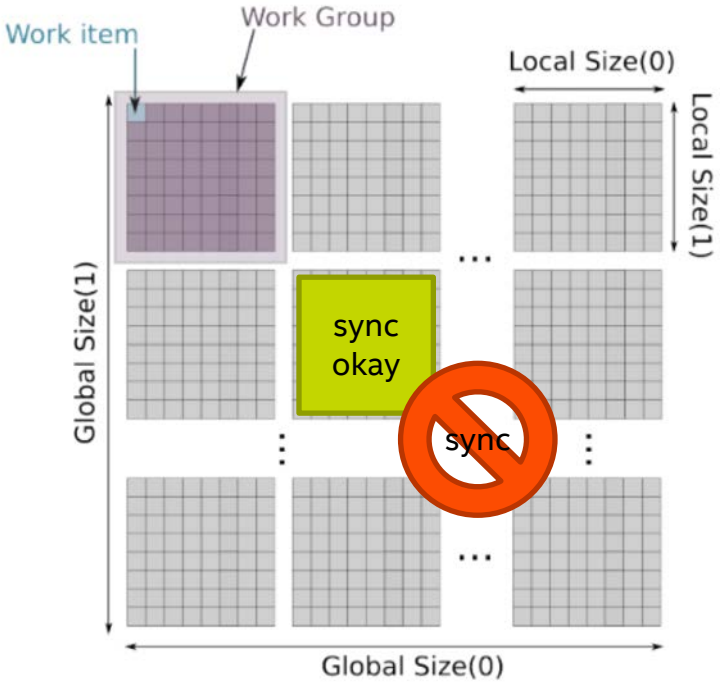
OpenMP TR6 is 671 pages (compare to MPI 3.1, which is 868).

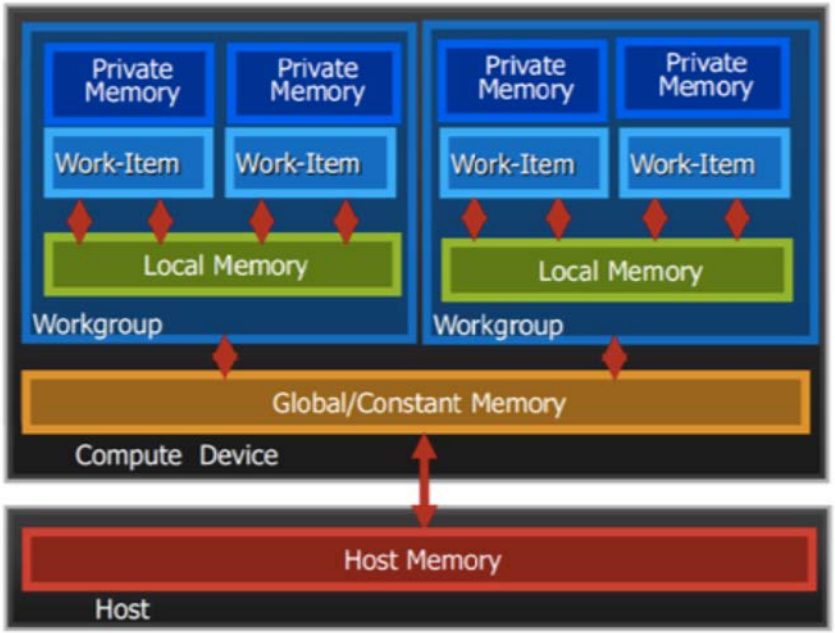
An Introduction to the OpenCL Programming Model

Jonathan Tompson*
NYU: Media Research Lab

Kristofer Schlachter†
NYU: Media Research Lab

OpenCL 2 is a bit more complicated, but doesn't change the execution model.





Simple data parallelism

```
// OpenMP
#pragma omp parallel for simd
for (size_t i=0; i<length; i++) {
      A[i] += B[i] + scalar * C[i];
}
```

We use std::vector unless otherwise noted.

Fun fact: STL containers cannot support NUMA in their current definition. This must be fixed to address NUMA before we can consider accelerator data.

```
// OpenCL
__kernel void axpy(int length, double scalar, __global double * A ...) {
   int i = get_global_id(0);
   if (i<length) {
      A[i] += B[i] + scalar * C[i];
   }
}
```

The host C/C++ code required to invoke OpenCL kernels is nontrivial, which discourages usage, although it is essential for runtime portability.

STL – sequential and parallel

```
auto b = std::begin(range);
auto e = std::end(range);

// Sequential STL
std::for_each( b, e, [&] (size_t i) {
      A[i] += B[i] + scalar * C[i];
});

// Parallel STL: par_unseq = threads+vector
std::for_each( pstl::execution::par_unseq, b, e, [&] (size_t i) {
      A[i] += B[i] + scalar * C[i];
});
```

We capture by reference because that is required to modify A when using an STL vector. Value capture works with raw pointers.

Parallel STL with ranges TS is in the works, which will eliminate b and e.

TBB

```
// affinity control
tbb::static_partitioner tbb_partitioner;

// specify a block size rather than relying on recursive bisection
tbb::blocked_range<size_t> range(0, length);

tbb::parallel_for( range, [&](decltype(range)& r) {
      for (auto i=r.begin(); i!=r.end(); ++i ) {
            A[i] += B[i] + scalar * C[i];
      }
}, tbb_partitioner);
```

Note that *TBB currently exposes the inner loop(s) explicitly*, which gives the programmer control over SIMD. The inner loop can be hidden (and is in tbb::for_each).

KOKKOS and RAJA

```
Kokkos::View<double*, Kokkos::OpenMP> A("A", length);
Kokkos::parallel_for( length, KOKKOS_LAMBDA(const size_t i) {
      A[i] += B[i] + scalar * C[i];
});

typedef RAJA::omp_parallel_for_exec thread_exec;
RAJA::forall<thread_exec>(0, length, [&] (RAJA::Index_type i) {
      A[i] += B[i] + scalar * C[i];
});
```

KOKKOS_LAMBDA enforces value capture and requirements of device compilation. This is unnecessary for CPU execution. RAJA has something similar for GPUs.

SYCL

SYCL is based on the OpenCL 1.2 specification, which is limiting. This is an area of active development.

```
sycl::cpu_selector device_selector; // host, cpu, gpu (default), acc
sycl::queue q(device_selector);
sycl::buffer<double> d_A { h_A.data(), h_A.size() };
..
q.submit([&](sycl::handler& h) {
      auto A = d_A.get_access<sycl::access::mode::read_write>(h);
      ..
      h.parallel_for<class foo>(sycl::range<1>{length}, [=] (sycl::item<1> i) {
            A[i] += B[i] + scalar * C[i];
      });
}); q.wait();
```

TBB (matrix reduction)

```
tbb::blocked_range2d<int> range(0, order, tile_size, 0, order, tile_size);

norm2 = tbb::parallel_reduce( range, double(0),
                        [&](decltype(range)& r, double temp) -> double {
                           for (auto i=r.rows().begin(); i!=r.rows().end(); ++i ) {
                              for (auto j=r.cols().begin(); j!=r.cols().end(); ++j ) {
                                 temp += std::fabs(A[i*order+j]);
                              }
                           }
                           return temp;
                        },
                        [] (const double x1, const double x2) { return x1+x2; },
                        tbb_partitioner );
```

KOKKOS (matrix reduction)

```
double norm2(0);
Kokkos::parallel_reduce(   team_policy(order, Kokkos::AUTO),
                           KOKKOS_LAMBDA(const member_type & teamMember,
                           double & update) {
      const int i = teamMember.league_rank();
      double temp(0);
      Kokkos::parallel_reduce(   Kokkos::TeamThreadRange(teamMember, order),
                                 [&](const int j, double & inner) {
            inner += std::fabs(A(i,j));
      }, temp);
      Kokkos::single( Kokkos::PerTeam( teamMember ), [&] () {
            update += temp;
      });
}, norm2);
```

RAJA (matrix reduction)

```
template <typename loop_policy, typename reduce_policy>
double Norm2(int order, std::vector<double> & A)
{
        // seq_exec disables vectorization in the inner loop
        typedef RAJA::NestedPolicy<RAJA::ExecList<loop_policy, RAJA::seq_exec>> exec_policy;

        RAJA::ReduceSum<reduce_policy, double> abserr(0.0);

        RAJA::forallN<exec_policy>( range(0, order), range(0, order), [=,&A](indx i, indx j) {
                norm2 += std::fabs(A[i*order+j]);
        });
        return norm2;
}

auto norm2 = Norm2<RAJA::omp_parallel_for_exec, RAJA::omp_reduce>(order,B);
```

Advanced RAJA

Example RAJA Execution Policy to apply

```
NestedPolicy<
 ExecList<
  seq_exec, seq_exec,
  omp_for_nowait_exec, simd_exec>,
 OMP_Parallel<
  Tile<
   TileList<
    tile_none, tile_none,
    tile_none, tile_fixed<512>>,
   Permute<PERM_JIKL>
  >
 >
>
```

Basic loop implementation

```
for d in range(0,dom<IDirection>(id)):
  for nm in range(0,dom<IMoment>(id)):
    for g in range(0,dom<IGroup>(id)):
      for z in range(0,dom<IZone>(id)):
```

Nested Policy applied to loop

```
#pragma omp parallel
for z2 in range(0,dom<IZone>(id),512):
  for d in range(0,dom<IDirection>(id)):
    for nm in range(0,dom<IMoment>(id)):
      #pragma omp for nowait
      for g in range(0,dom<IGroup>(id)):
        for z in range(z2,z2+512):
```

Model	for	for^N	reduce	scan	Hierarchy/Composition
TBB::parallel	Y	Y	Y	Y	Threads
C++17 PSTL	Y	N^	Y	Y	Threads+SIMD
RAJA	Y	Y	Y	Y	Threads+SIMD; CUDA
KOKKOS	Y	Y	Y	Y	Team+Thread+SIMD
Boost.Compute	Y	N*^	Y	Y	N
SYCL	Y	3	N	N	N
OpenCL 1.x	Y	3	N	N	N
OpenMP 5	Y	Y	Y	5	Y**

* Boost.Compute supports embedded OpenCL, which in turn exposes 3D loop nests.

** OpenMP nested parallelism is unpleasant. You can nest “parallel for” or switch paradigms to “taskloop” and give up on accelerator support.

^ One can always implement a collapsed N-d loop but that adds div/mod to loop body.

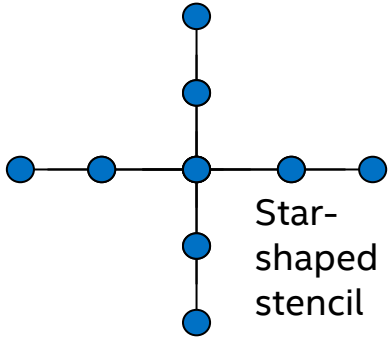
HPC-like vs STL-like vs OpenCL-like

- TBB — HPC-like
  - Nested, blocked forall w/ affinity control and load-balancing
- RAJA
  - Nested, blocked, permuted forall w/ fine-grain policy control.
- KOKKOS
  - Nested, blocked, permuted forall.

- C++17 (parallel STL) — STL-like
  - Parallel STL evolving towards GPU etc.
- Boost.Compute
  - Effectively parallel STL over OpenCL.
- SYCL — OpenCL-like
  - OpenCL execution model
  - Parallel STL over SYCL exists*.

The HPC-like models capture the popular OpenMP idioms while hiding complexity.

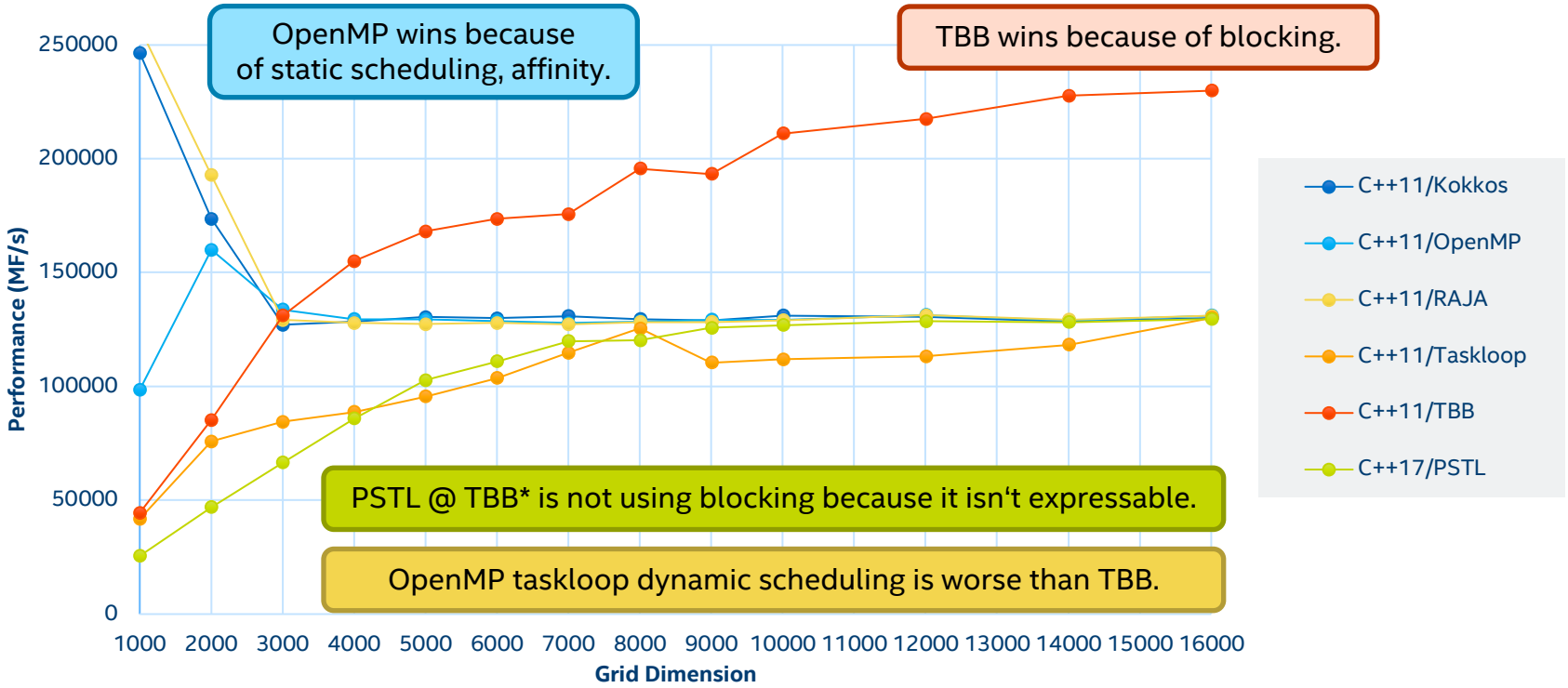
* https://github.com/KhronosGroup/SyclParallelSTL



PERFORMANCE EXPERIMENTS

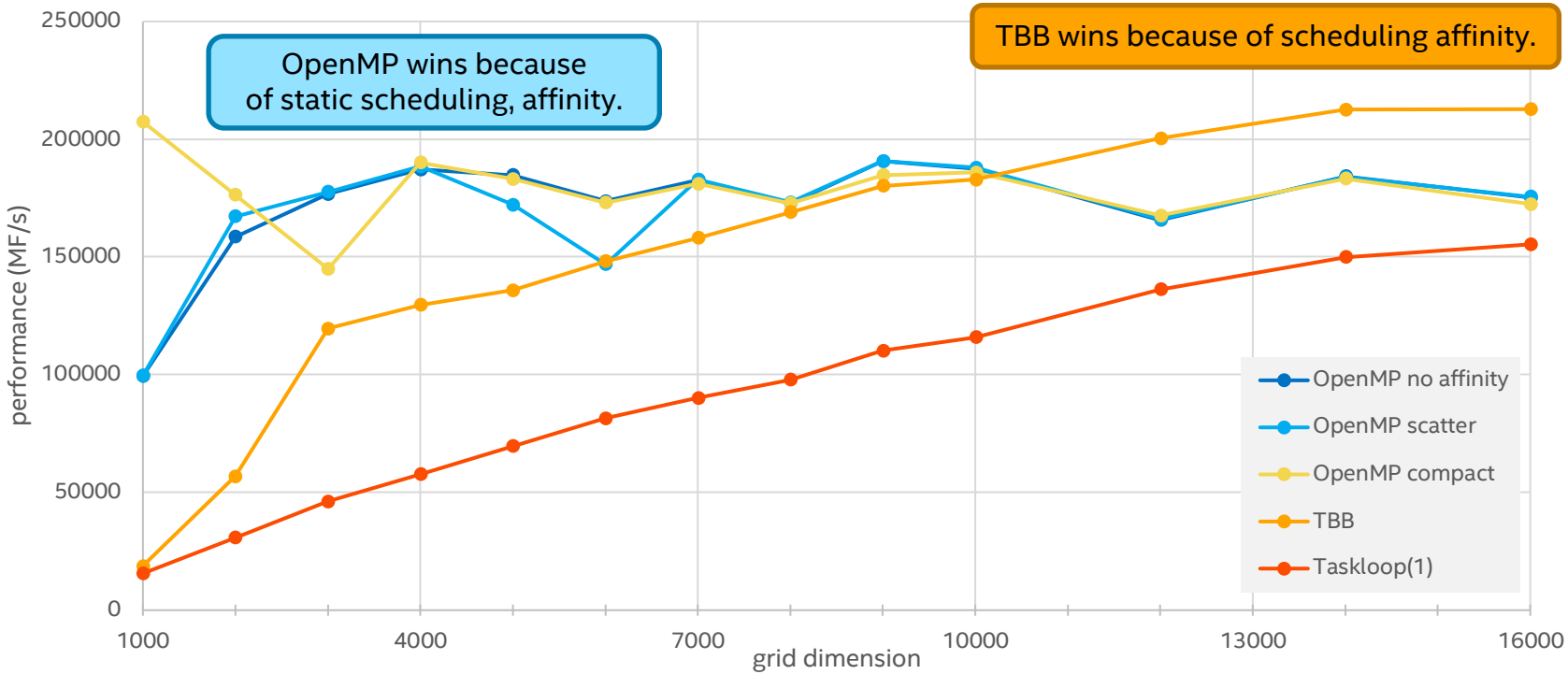
https://github.com/ParRes/Kernels/tree/master/Cxx11

PRK stencil: C++ implementations on KNL

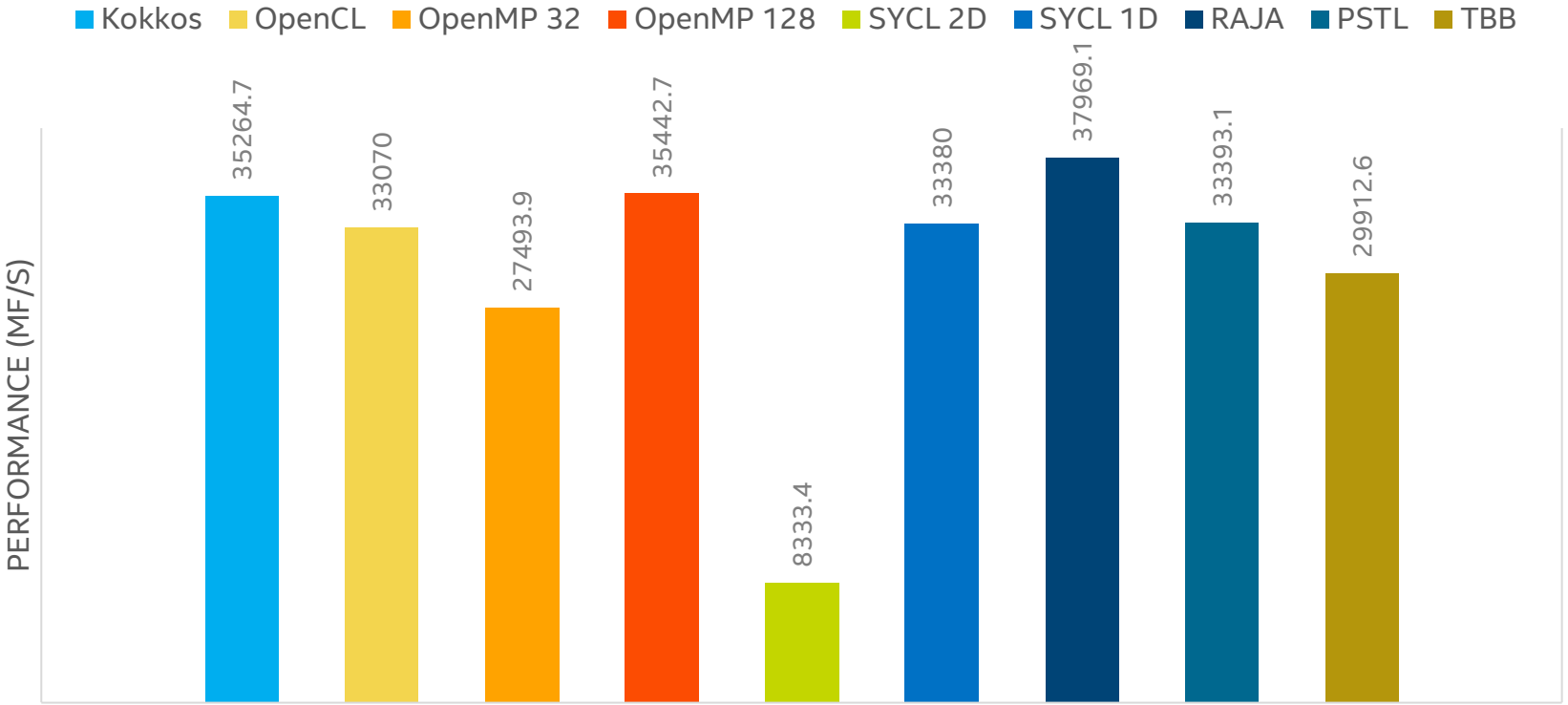


https://github.com/intel/parallelstl

Improved PRK stencil on KNL (MCDRAM flat)



PRK stencil - Intel Core i7-5960X



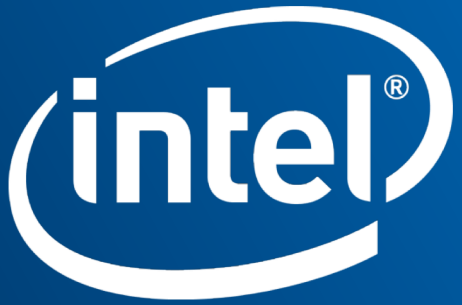
SYCL indexing makes a huge difference on performance.

Summary

- KOKKOS and RAJA are zero-overhead abstractions on top of OpenMP.
- KOKKOS and RAJA provide a path to portability across architectures.
- TBB offers productive performance advantages relative to OpenMP on CPUs.
- PSTL (based on TBB in Intel's implementation) works well on CPUs but is limited by STL semantics.
- SYCL unifies STL and single-source with OpenCL execution model.
- PSTL portability requires evolution of C++ towards HPX, Thrust…

References

- R. F. Van der Wijngaart, A. Kayi, J. R. Hammond, G. Jost, T. St. John, S. Sridharan, T. G. Mattson, J. Abercrombie, and J. Nelson. ISC 2016. *Comparing runtime systems with exascale ambitions using the Parallel Research Kernels.*
- *E.* Georganas, R. F. Van der Wijngaart and T. G. Mattson. IPDPS 2016. *Design and Implementation of a Parallel Research Kernel for Assessing Dynamic Load-Balancing Capabilities.*
- R. F. Van der Wijngaart, S. Sridharan, A. Kayi, G. Jost, J. Hammond, T. Mattson, and J. Nelson. PGAS 2015. *Using the Parallel Research Kernels to study PGAS models.*
- R. F. Van der Wijngaart and T. G. Mattson. HPEC 2014. *The Parallel Research Kernels.*


intel®

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

PARALLEL RESEARCH KERNELS

Programming model evaluation

Standard methods

- NAS Parallel Benchmarks
- Mini Applications
  (e.g. Mantevo, LULESH)
- HPC Challenge

There are numerous examples of these on record, covering a wide range of programming models, but is source available and curated*?

What is measured?

- Productivity (?), elegance (?)
- Implementation quality
  (runtime or application)
- Asynchrony/overlap
- Semantics:
  - Automatic load-balancing (AMR)
  - Atomics (GUPS)
  - Two-sided vs. one-sided, collectives

* PRK curation is currently active. - no future commitment, but you can GitHub fork and carry the torch.

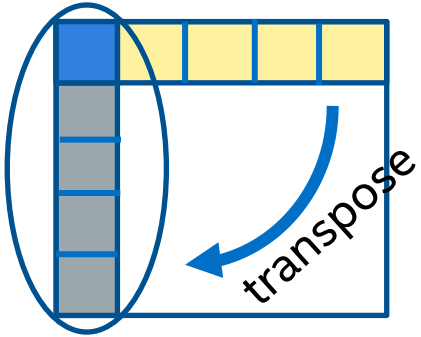
Goals of the Parallel Research Kernels

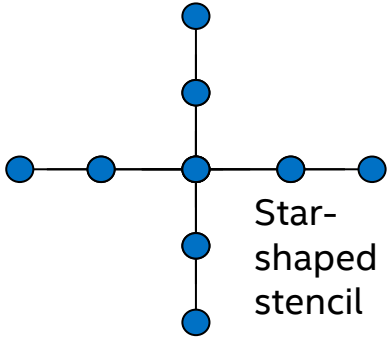
1. **Universality**: Cover broad range of performance critical application patterns.
2. **Simplicity**: Concise pencil-and-paper definition and transparent C reference implementation. *No domain knowledge required.*
3. **Portability**: Should be implementable in any sufficiently general programming model.
4. **Extensibility**: Parameterized to run at any scale. Other knobs to adjust problem or algorithm included.
5. **Verifiability**: Automated correctness checking and built-in performance metric evaluation.

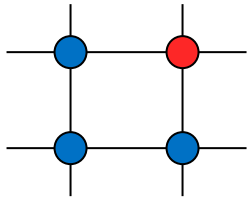
~~6. Hardware benchmark~~: No! Use HPCChallenge, Xyz500, etc. for this.

Outline of PRK Suite

- ***Dense matrix transpose***
- Synchronization: global
- ***Synchronization: point to point***
- Scaled vector addition
- Atomic reference counting
- Vector reduction
- Sparse matrix-vector multiplication
- Random access update
- ***Stencil computation***
- Dense matrix-matrix multiplication
- Branch
- Particle-in-cell (new)
- (more in progress)







$$A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}$$

Static kernels

PRK implementations - part 1

- Serial (C89)
- OpenMP (C89)
- MPI1 – MPI two-sided
  - Also support FG-MPI (UBC) and AMPI (UIUC)
- MPIOMP – MPI two-sided with local OpenMP
- MPISHM – MPI two-sided with MPI-3 shared-memory
- MPIRMA – MPI one-sided communication (multiple flavors)
- SHMEM (C89)
- UPC
- Grappa (C++)
- Charm++ (C++)
- Chapel
- Fortran 2008 coarrays

Serial C, OpenMP, and MPI support most of the PRKs. **Synch_p2p**, **Stencil** and **Transpose** are primary targets for evaluation.

In progress (?):
Legion (Stanford)
HPX (LSU & IU)
OCR (Rice/Intel)

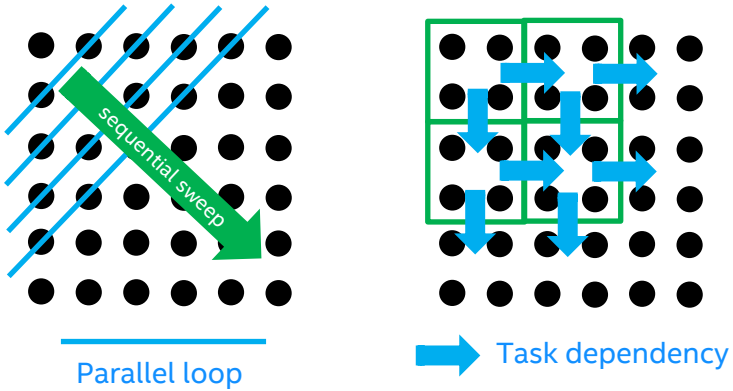
PRK implementations - part 2

Stencil and **Transpose** are both data-parallel and are implemented in a wide range of models using parallel loops using TOVI (Thread Outer, Vector Inner).

- Fortran, C, C++
  - Serial
  - OpenMP
  - OpenMP taskloop
  - OpenMP target
- Fortran
  - pretty (intrinsics/colon notation)
  - OpenACC
- C++
  - (P)STL (C++17)
  - TBB
  - RAJA
  - Kokkos
  - OpenCL
  - Async (transpose only)
  - CUDA (transpose only)
- C, C++
  - Cilk
  - Threads (transpose only)

PRK implementations - part 3

Synch_p2p exemplifies the wavefront pattern, which is amenable to task parallelism, inefficient data parallelism, etc.



Inner-loop / anti-diagonals

- All: Serial, OpenMP
- Fortran: OpenACC
- C++: TBB, OpenCL, RAJA

Task-based

- All: OpenMP tasks
- C++: TBB (in-progress)

Wavefront loops

- All: OpenMP doacross (bad impls.)

All = C, C++, Fortran

PRK implementations - part 4

- Python
  - Native
  - Numpy (intrinsics or colon notation)
- Julia
  - Loops
- Octave (Matlab)
  - Loops
  - Pretty (intrinsics or colon notation)
- Rust

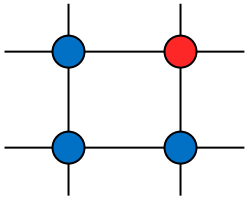
Both Python and Julia support ParallelAccelerator, but this is not implemented yet.

Rust is a hard language to learn. The PRK implementation is the first and only Rust code I've ever written.

Synch point-to-point

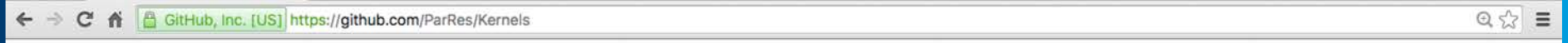
```
for i in range(1,m):
      for j in range(1,n):
            grid[i][j]     = grid[i-1][j]
                           + grid[i][j-1]
                           - grid[i-1][j-1]

grid[0][0] = -grid[m-1][n-1]
```



$A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}$

- Proxy for discrete ordinates neutron transport (e.g. PARTSN); much simpler than SNAP proxy.
- Proxy for dynamic programming, which is used in sequence alignment (i.e. bioinformatics).
- Wraparound to create dependency between iterations.



ParRes / **Kernels**

Watch 12 | Star 38 | Fork 20

Code | Issues 24 | Pull requests 3 | Wiki | Pulse | Graphs

This is a set of simple programs that can be used to explore the features of a parallel platform.
https://groups.google.com/forum/#!forum/parallel-research-kernels

1,252 commits | **11** branches | **5** releases | **10** contributors

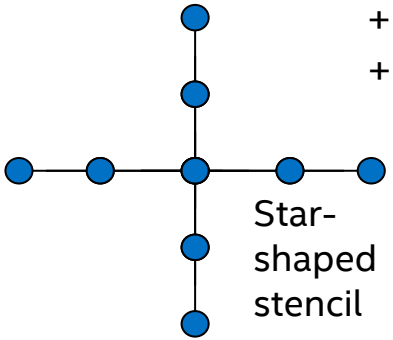
Branch: **master** | New pull request | Find file | Clone or download

jeffhammond Merge pull request #111 from jeffhammond/travis-opencoarrays-2 ... Latest commit 54da8be 12 days ago

AMPI	more missing -lm	22 days ago
CHARM++	more missing -lm	22 days ago
FG_MPI	add more missing libm for sqrt and ceil	21 days ago
FORTRAN	Merge pull request #106 from jeffhammond/julia	13 days ago
GRAPPA	Adding debug info to GRAPPA/{Stencil,Transpose}	2 months ago
JULIA	remove temp files [ci skip]	13 days ago
LEGION	Adding OpenMP version of PIC, reverting OpenMP Transpose to "old" ver…	a month ago
MPI1	more missing -lm	22 days ago
MPIOPENMP	add more missing libm for sqrt and ceil	21 days ago
MPIRMA	sqrt() require libm - resolves #102	22 days ago

Stencil

```
B[2:n-2,2:n-2] += W[2,2] * A[2:n-2,2:n-2]
                + W[2,0] * A[2:n-2,0:n-4]
                + W[2,1] * A[2:n-2,1:n-3]
                + W[2,3] * A[2:n-2,3:n-1]
                + W[2,4] * A[2:n-2,4:n-0]
                + W[0,2] * A[0:n-4,2:n-2]
                + W[1,2] * A[1:n-3,2:n-2]
                + W[3,2] * A[3:n-1,2:n-2]
                + W[4,2] * A[4:n-0,2:n-2]
```

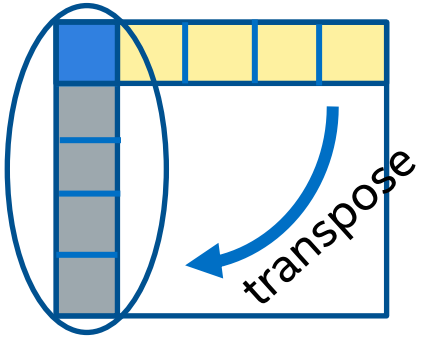


Star-shaped stencil

- Proxy for structured mesh codes. 2D stencil to emphasize non-compute.
- Supports arbitrary radius star and square stencils via code generator for C11 and C++ models, which was inspired by OpenCL.

Transpose

```
for i in range(order):
    for j in range(order):
        B[i][j] += A[j][i]
        A[j][i] += 1.0
```



- Proxy for 3D FFT, bucket sort…
- Local transpose of square tiles supports blocking to reduce TLB pressure.