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

Jeff Hammond, Exascale Co-Design Group
Tim Mattson, Parallel Computing Lab
Intel Corporation

13 May 2019

Acknowledgements: **Rob van der Wijngaart**, Alex Duran, Jim Cownie, Alexey Kukanov, Pablo Reble, Xinmin Tian, Martyn Corden, Tom Scoglund and the rest of the RAJA team at LLNL, CodePlay SYCL team, …

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.

Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. . Performance varies depending on system configuration. **No computer system can be absolutely secure.** Check with your system manufacturer or retailer or learn more at intel.com/performance/datacenter.

Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

The cost reduction scenarios described are intended to enable you to get a better understanding of how the purchase of a given Intel based product, combined with a number of situation-specific variables, might affect future costs and savings. Circumstances will vary and there may be unaccounted-for costs related to the use and deployment of a given product. Nothing in this document should be interpreted as either a promise of or contract for a given level of costs or cost reduction.

Intel processors of the same SKU may vary in frequency or power as a result of natural variability in the production process.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

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.

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. 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. For more information go to www.intel.com/benchmarks

*Other names and brands may be claimed as the property of others.

Additional Disclaimer

I am not an official spokesman for any Intel products. 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.

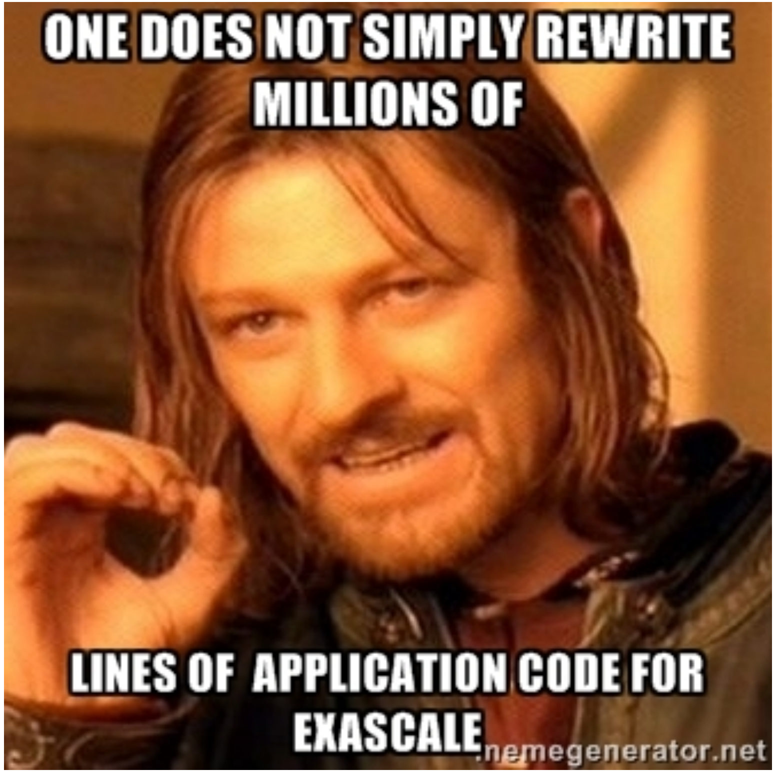
You may or may not be able to reproduce any performance numbers I report, but the code is on GitHub* and I will provide anything else you need to attempt to reproduce my results.

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

HPC software design challenges (2014)

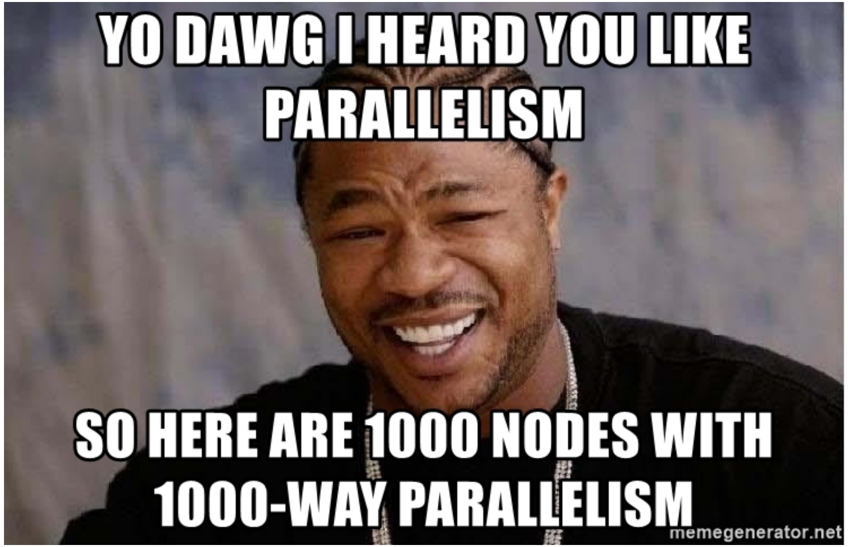
- To MPI or not to MPI…
- One-sided vs. two-sided?
- Does your MPI/PGAS need a +X?
- Static vs. dynamic execution model?
- What synchronization motifs maximize performance across scales?

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



HPC software design challenges (2018)

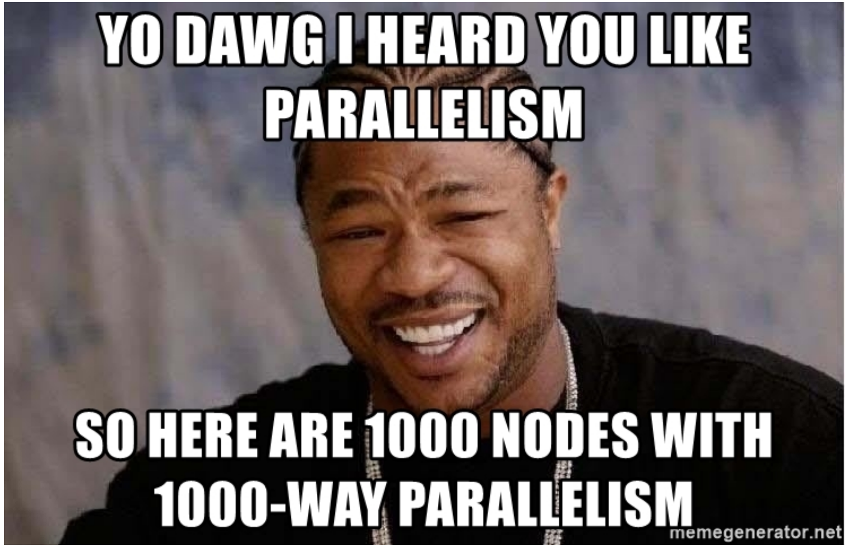
- Intranode parallelism is growing much fast than internode…
- Intranode parallelism is far more diverse than internode parallelism.
  - After ~20 years, internode behavior is converged to some subset of MPI-3.
  - Big Cores, Little Cores, GPU, FPGA all require (very) different programming models.



How do we maximize productivity+performance+portability?

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 *measure* productivity+performance+portability?

PARALLEL RESEARCH KERNELS

Jeff Hammond
Exascale Co-Design Group

Programming model evaluation

Standard methods:

- NAS Parallel Benchmarks
- Mini/Proxy Applications
- 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

Benchmark Suite Scorecard (EMBRACE 2017)

	HPC user interest	clear paper & pencil	fast reference code	clear reference code	forum for comparison	productivity framework	prescribed approach	open competition	historical continuum	apples-to-apples	self-serve comparisons	self-run entries
NPB	✓	~	✓	~	✗	✗	✗					
HPCC	~	~	✓	✗	✓	✓	~	✓	✗	✗	✗	✗
DOE PROXY APPS	✓	~	✓	~	✗	✗	~					
CLBG	✗	✗	✓	✓	✓	✓	✓	~	✓	✓	✓	✓
PRK	~	✓	✓	?	~	✗	✓	✓				✓

Jeff Hammond
Exascale Co-Design Group

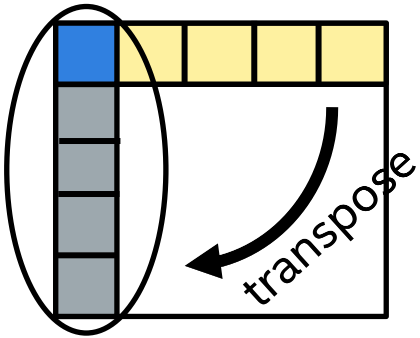
https://chapel-lang.org/presentations/Chamberlain-Dagstuhl-presented.pdf

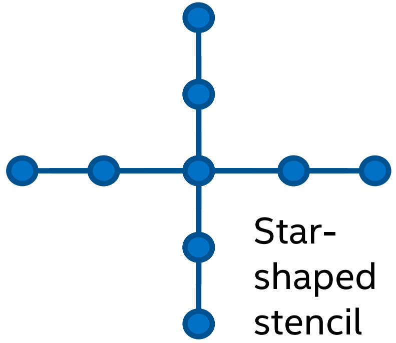
Goals of the Parallel Research Kernels

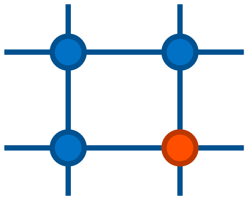
1. Universality: Cover broad range of performance critical application patterns.
2. Simplicity: Concise pencil-and-paper definition and transparent reference implementation. No domain knowledge required.
3. Portability: Should be implementable in any sufficiently general programming model.
4. Extensibility: 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

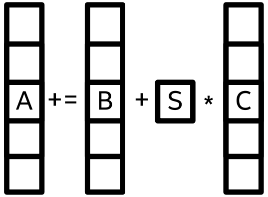
- ***Dense matrix transpose***
- Synchronization: global
- ***Synchronization: point to point***
- ***Scaled vector addition***
- Atomic reference counting
- Vector reduction
- Sparse matrix-vector multiplication
- Random access update
- ***Stencil computation***
- Dense matrix-matrix multiplication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







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



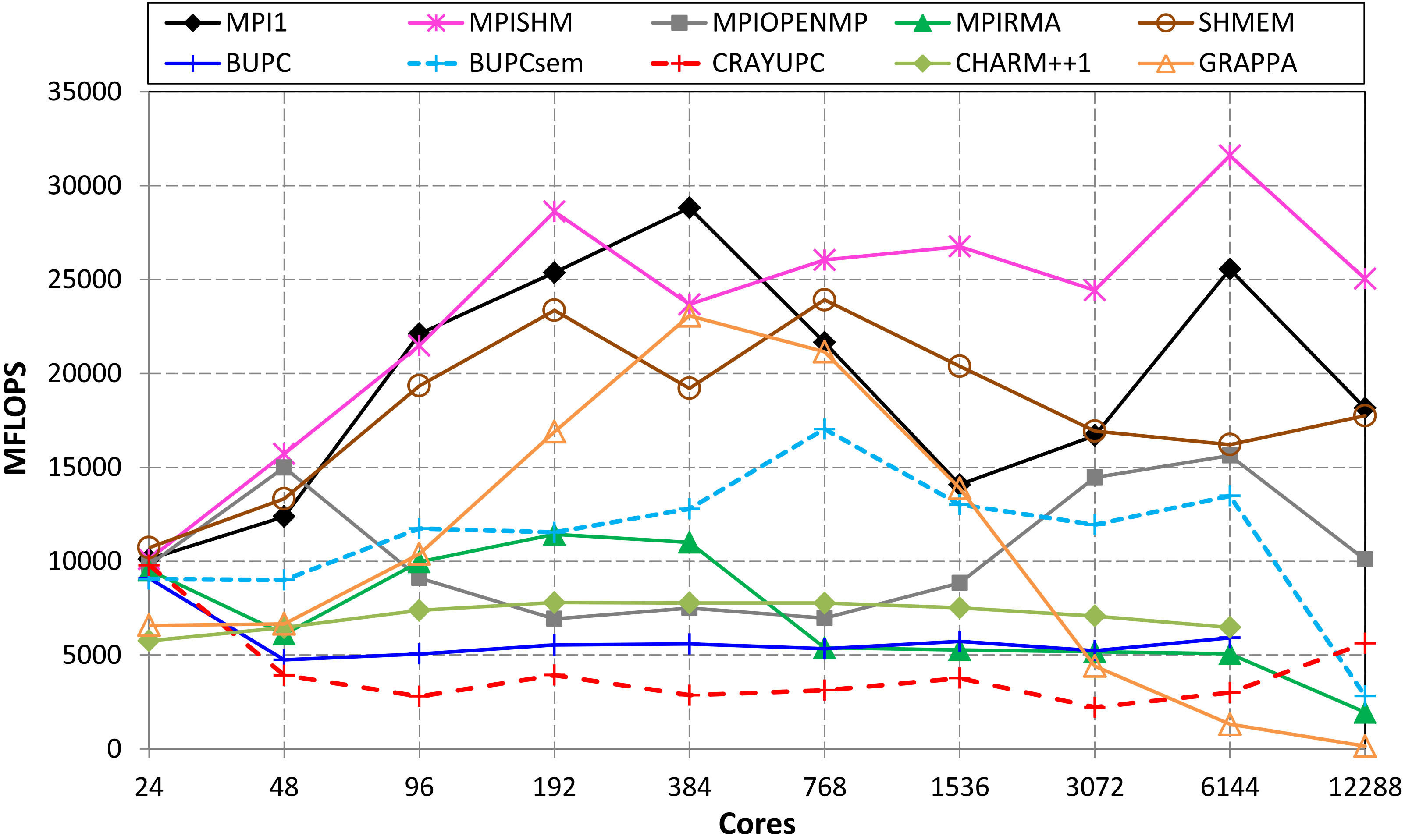
Static kernels

Language	Seq.	OpenMP	MPI	PGAS	Threads	Others?
C89	√	√	Many	SHMEM		
C99/C11	√	√√√		UPC	√	~~Cilk~~, ISPC
C++17	√	√√√		Grappa	√	Kokkos, RAJA, TBB, PSTL, SYCL, OpenCL, CUDA…
Fortran	√	√√√		coarrays		“pretty”, OpenACC
Python	√					Numpy
Chapel	√			√		

√√√ = Traditional, task-based, and target are implemented identically in Fortran, C and C++.

Additional language support includes Rust, Julia, and Matlab/Octave.

ISC 2016



https://link.springer.com/chapter/10.1007%2F978-3-319-41321-1_17

ParRes / Kernels

https://github.com/ParRes/Kernels

Unwatch 24 | Unstar 142 | Fork 40

Code | Issues 27 | Pull requests 3 | Projects 0 | Wiki | Insights | Settings

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

parallel-programming parallel-computing c c-plus-plus mpi fortran2008 python3 julia pgas openmp shmem coarray-fortran travis-ci charmplusplus threading tbb kokkos opencl sycl boost

Manage topics

2,822 commits | 11 branches | 6 releases | 19 contributors | View license

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

jeffhammond avoid overflow — Latest commit 30a2c6f 19 days ago

.github	try out issue/PR templates	a year ago
AMPI	avoid overflow	16 days ago
C1z	OpenCL: add No Device errors (#373)	29 days ago
CHARM++	avoid overflow	16 days ago
Cxx11	do not incorrectly declare non-read-only buffers as read-only	19 days ago

ParRes / Kernels build failing https://travis-ci.org/ParRes/Kernels

Current Branches Build History Pull Requests More options

master CRON avoid overflow #1410 failed Restart build

Commit 30a2c6f

Branch master

Jeff Hammond authored and committed

Ran for 1 hr 44 min 57 sec

Total time 4 hrs 45 min 26 sec

6 days ago

Build jobs View config

# 1410.1	Compiler: gcc C++	PRK_TARGET=allserial	2 min 1 sec
# 1410.2	Compiler: clang C++	PRK_TARGET=allserial	5 min 33 sec
# 1410.3	Compiler: gcc C++	PRK_TARGET=allc1z	6 min 53 sec
# 1410.4	Compiler: clang C++	PRK_TARGET=allc1z	1 min 9 sec
# 1410.5	Compiler: gcc C++	PRK_TARGET=allcxx	8 min 13 sec
# 1410.6	Compiler: clang C++	PRK_TARGET=allcxx	6 min 35 sec
# 1410.7	Compiler: clang C++	PRK_TARGET=allpython	7 min 9 sec
# 1410.8	Compiler: clang C++	PRK_TARGET=alljulia	5 min 16 sec
# 1410.9	Compiler: gcc C++	PRK_TARGET=allopenmp	3 min 7 sec
# 1410.10	Compiler: gcc C++	PRK_TARGET=allfortran	14 min 44 sec
# 1410.11	Compiler: gcc C++	PRK_TARGET=allmpi	16 min 20 sec
# 1410.12	Compiler: gcc C++	PRK_TARGET=allshmem	4 min 59 sec

ParRes /

Current Branches Bu

More options

master CRON a

Commit 30a2c

Branch master

Jeff Hammond aut

Restart build

Bu

```
+SYCLDIR=/Users/travis/build/ParRes/Kernels/PRK-deps/triSYCL
+'[' clang = clang ']'
+echo 'SYCLCXX=/usr/local/opt/llvm/bin/clang++ -pthread -std=c++17'
+echo 'SYCLFLAG=-DUSE_SYCL -I/Users/travis/build/ParRes/Kernels/PRK-deps/triSYCL/include'
+make -C Cxx11 p2p-hyperplane-sycl stencil-sycl transpose-sycl nstream-sycl
/usr/local/opt/llvm/bin/clang++ -pthread -std=c++17 -DPRKVERSION="2.16" -DUSE_SYCL -
I/Users/travis/build/ParRes/Kernels/PRK-deps/triSYCL/include -DUSE_SYCL -DUSE_2D_INDEXING=0 -DUSE_RANGES_TS -
I/Users/travis/build/ParRes/Kernels/PRK-deps/range-v3/include -DUSE_RANGES p2p-hyperplane-sycl.cc -o p2p-hyperplane-sycl
In file included from p2p-hyperplane-sycl.cc:62:
In file included from /Users/travis/build/ParRes/Kernels/PRK-deps/triSYCL/include/CL/sycl.hpp:40:
In file included from /Users/travis/build/ParRes/Kernels/PRK-deps/triSYCL/include/CL/sycl/buffer.hpp:27:
In file included from /Users/travis/build/ParRes/Kernels/PRK-deps/triSYCL/include/CL/sycl/queue.hpp:30:
/Users/travis/build/ParRes/Kernels/PRK-deps/triSYCL/include/CL/sycl/property_list.hpp:63:1: error: call to unavailable
member function 'value': introduced in macOS 10.14
TRISYCL_PROPERTY_HAS_GET(queue, enable_profiling)
^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/Users/travis/build/ParRes/Kernels/PRK-deps/triSYCL/include/CL/sycl/property_list.hpp:60:22: note: expanded from macro
'TRISYCL_PROPERTY_HAS_GET'
    return prop_name.value();                                           \
           ~~~~~~~~~~^~~~~
/usr/local/opt/llvm/bin/../include/c++/v1/optional:938:33: note: candidate function has been explicitly made unavailable
    constexpr value_type const& value() const&
                                ^
/usr/local/opt/llvm/bin/../include/c++/v1/optional:947:27: note: candidate function not viable: 'this' argument has type
'const std::optional<property::queue::enable_profiling>', but method is not marked const
    constexpr value_type& value() &
                          ^
/usr/local/opt/llvm/bin/../include/c++/v1/optional:956:28: note: candidate function not viable: 'this' argument has type
'const std::optional<property::queue::enable_profiling>', but method is not marked const
    constexpr value_type&& value() &&
                           ^
/usr/local/opt/llvm/bin/../include/c++/v1/optional:965:34: note: candidate function not viable: no known conversion from
'const optional<...>' to 'const optional<...>' for object argument
    constexpr value_type const&& value() const&&
                                 ^
1 error generated.
make: *** [p2p-hyperplane-sycl] Error 1
```

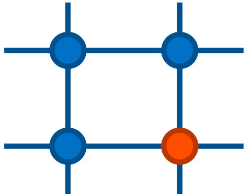
Top

# 1410.1	2 min 1 sec
# 1410.2	5 min 33 sec
# 1410.3	6 min 53 sec
# 1410.4	1 min 9 sec
# 1410.5	8 min 13 sec
# 1410.6	6 min 35 sec
# 1410.7	7 min 9 sec
# 1410.8	5 min 16 sec
# 1410.9	3 min 7 sec
# 1410.10	14 min 44 sec
# 1410.11	16 min 20 sec
# 1410.12	4 min 59 sec

Synch point-to-point

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

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

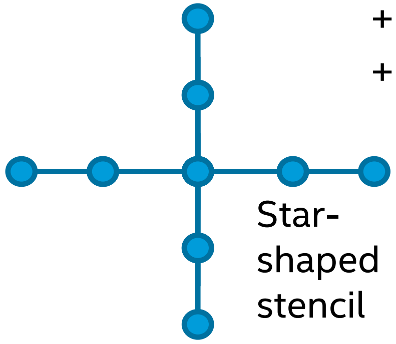


$A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}$

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

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]
```

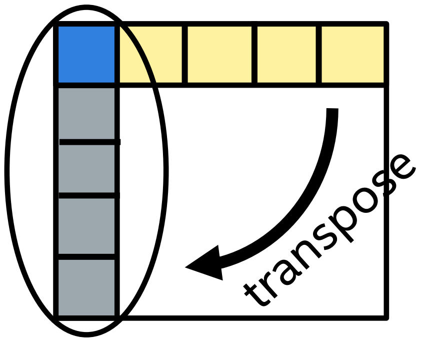


Star-shaped stencil

- 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.

C++ AND PARALLELISM

Jeff Hammond
Exascale Co-Design Group

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!

Overview of Parallel C++ models

- TBB (Intel OSS) - parallel threading abstraction for CPU architectures.
- 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/Umpire 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, …).

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	Group(+Subgroup)+Item
OpenCL	Y	3	N	N	Group+Item
OpenMP 5	Y	Y	Y	Y	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

HPC-like

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

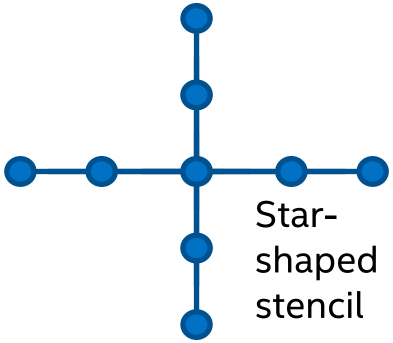
STL-like

- C++17 (parallel STL)
  - Parallel STL evolving towards GPU etc.
- Boost.Compute
  - Effectively parallel STL over OpenCL.

OpenCL-like

- SYCL
  - OpenCL execution model
  - Parallel STL over SYCL exists…

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



PERFORMANCE EXPERIMENTS

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

The performance data has been removed…

- The experimental results are meant to be illustrative of what can be learned from the PRKs. We encourage you to run your own experiments, since performance data tends to go stale rather quickly. Please email Jeff if you need any assistance with this task.
- The results I showed demonstrated the following:
    - TBB beats OpenMP for naïve usage because TBB parallel_for compels the user to block for cache, whereas OpenMP requires the user to implement it themselves.
    - Kokkos naturally handles NUMA-aware allocation, whereas STL containers do not. It's necessary to avoid the STL when NUMA-awareness is required.
    - Kokkos, RAJA, TBB, PSTL, OpenCL and SYCL all produce the same quality of results (i.e. performance) when the code is written the same way. There is no inherent advantage or disadvantage to any of these models from a performance perspective.

Summary

- Parallel C++ models effectively hide the complexity of underlying models like OpenMP and OpenCL without introducing any overhead (on CPUs).
- Implementation differences between OpenMP and TBB schedulers show places where OpenMP runtimes can be improved.
- PSTL (based on TBB in Intel's implementation) works well on CPUs but is limited by STL semantics. PSTL portability requires evolution of C++ towards HPX, Thrust…
- SYCL provides a modern C++ abstraction and single-source compilation on top the OpenCL execution model.
- GPU-oriented models lack (rely on external libraries for) important primitives.

Where do we go next?

- Continuously trying to keep up with RAJA and other moving targets…
- Evaluate performance on other platforms, particularly non-CPU ones.
- Performance optimization, particularly in stencil – how productive is tuning in different models?
- Write additional kernels:
    - Branch 2.0 (orient towards lane divergence, not branch predictor)
    - Reduce (different patters, variable implementation quality)
- Julia vs Python vs Octave doesn't matter to me but others care.

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 and T. G. Mattson. HPEC 2014. *The Parallel Research Kernels.*

intel®