

THE PARALLEL RESEARCH KERNELS AND THEIR USE IN CO-DESIGN

Jeff Hammond
NVIDIA Helsinki

Acknowledgements

- **Tim Mattson and Rob van der Wijngaart (Intel)**
- Apo Kayi, Gabi Jost, Tom St John, Srinivas Sridharan, Kiran Pamnany, Alex Duran, Alexey Kukanov, Pablo Reble, Xinmin Tian, Jim Cownie, Martyn Corden, Steve Lionel, James Brodman (Intel)
- John Abercrombie, Jacob Nelson (U. Washington)
- Wonchan Lee (Stanford)
- Yijian (Tim) Hu, (Georgia Tech)
- Lisandro Dalcin and Marcin Rogowski (KAUST)
- Brad Chamberlain (Cray)
- Christian Trott (Sandia), Tom Scogland (LLNL)
- Alessandro Fanfarillo (NCAR)
- Thomas Hayward-Schneider (MPG)
- Sajid Ali (NYU)
- Cedric Augonnet (NVIDIA)
- Carsten Bauer (NHR)
- Many others, not listed here.

PIRIK

PARALLEL RESEARCH KERNELS

<https://github.com/ParRes/Kernels/> has all the goods

Project History

- Created by Tim Mattson a long time ago to understand CPU architecture.
- C89-based MPI-1 and OpenMP-3 ports by Tim and Rob.
- Used for Intel exascale software study in 2014-2016.
 - UPC is C99, Charm++ and Grappa are C++ - we should move beyond C89.
 - Chapel vs C89/MPI is not a useful comparison (hence “pretty” versions).
- Extended to C++ and Fortran (2016-2018) with offload models because exascale was always going to be heterogeneous.
- Non-HPC language ports started as a hobby project (2017-2020).
- GPU studies (2019-present).

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)
 - Two-sided vs. one-sided, collectives

“You can't manage what you can't measure”
- Peter Drucker

Goals of the Parallel Research Kernels

Universal: Cover broad range of performance critical application patterns.

Simple: Concise pencil-and-paper definition and transparent reference implementation. No domain knowledge required.

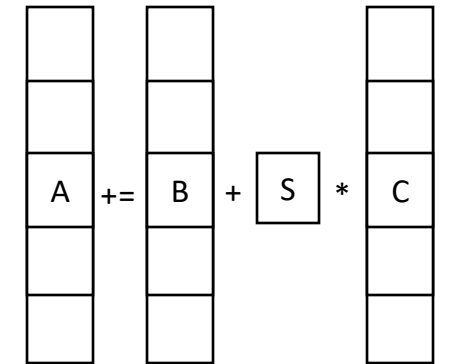
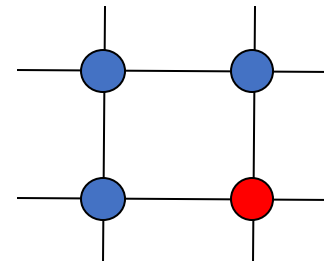
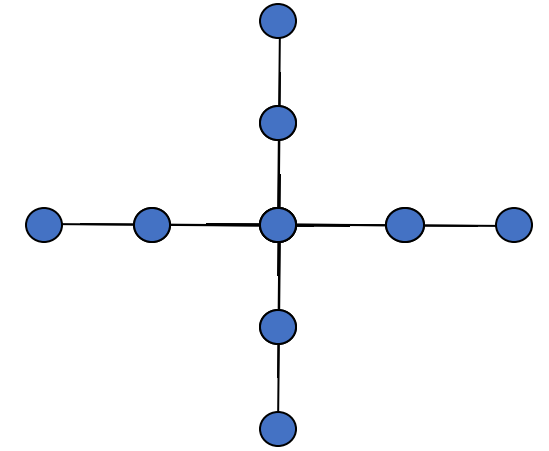
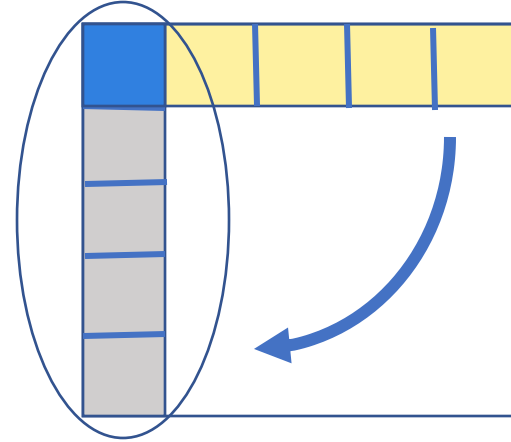
Portable: Should be implementable in any sufficiently general programming model.

Extensible: Parameterized to run at any scale. Other knobs to adjust problem or algorithm included.

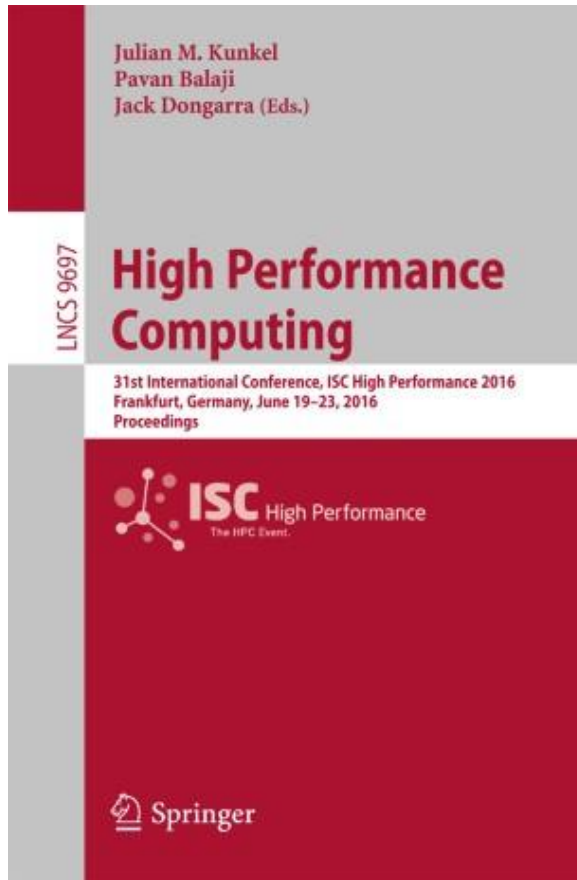
Verifiable: Automated correctness checking and built-in performance metric evaluation.

Outline of PRK Suite

- **Dense matrix transpose**
- Synchronization: global
- **Synchronization: point to point (p2p)**
- **Scaled vector addition (nstream)**
- Vector reduction
- Sparse matrix-vector multiplication
- Random access update
- **Stencil computation**
- Dense matrix-matrix multiplication
- Branch
- Particle-in-cell



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



Comparing runtime systems with exascale ambitions using the Parallel Research Kernels

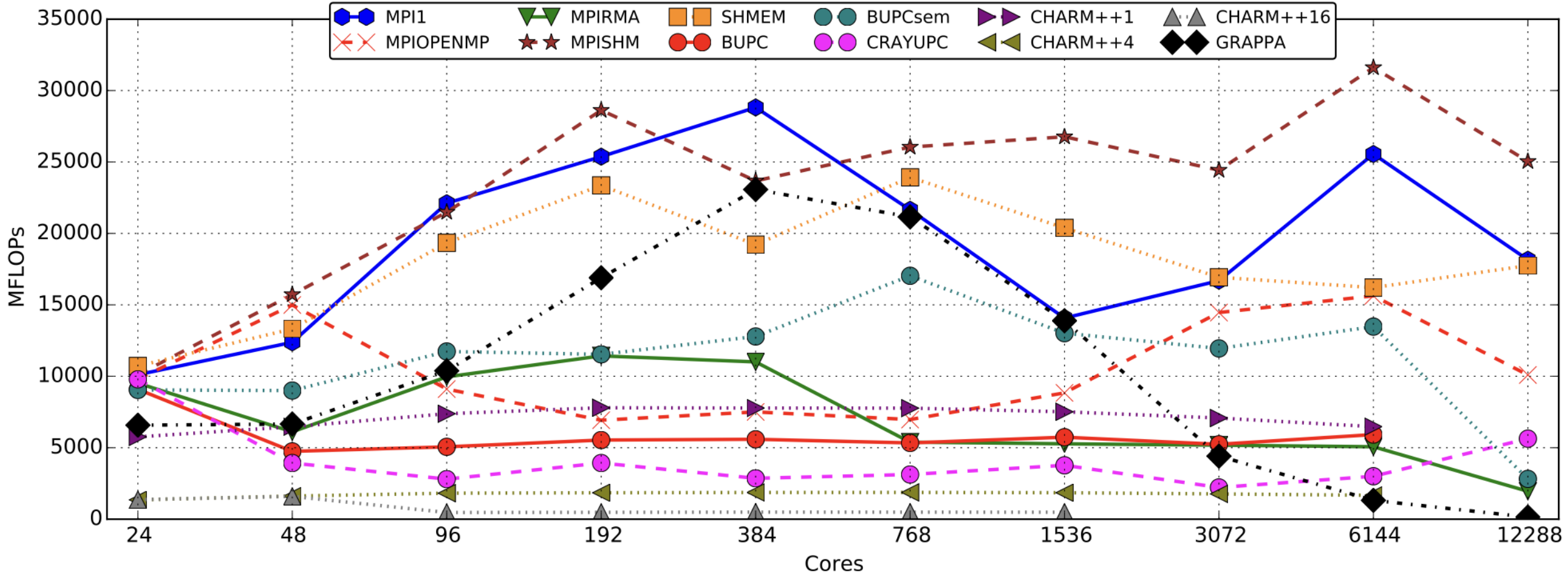
R. F. Van der Wijngaart¹, A. Kayi¹, J. R. Hammond¹, G. Jost¹, T. St. John¹,
S. Sridharan¹, T. G. Mattson¹, J. Abercrombie², and J. Nelson²

¹ Intel Corporation, Hillsboro, Oregon, USA.

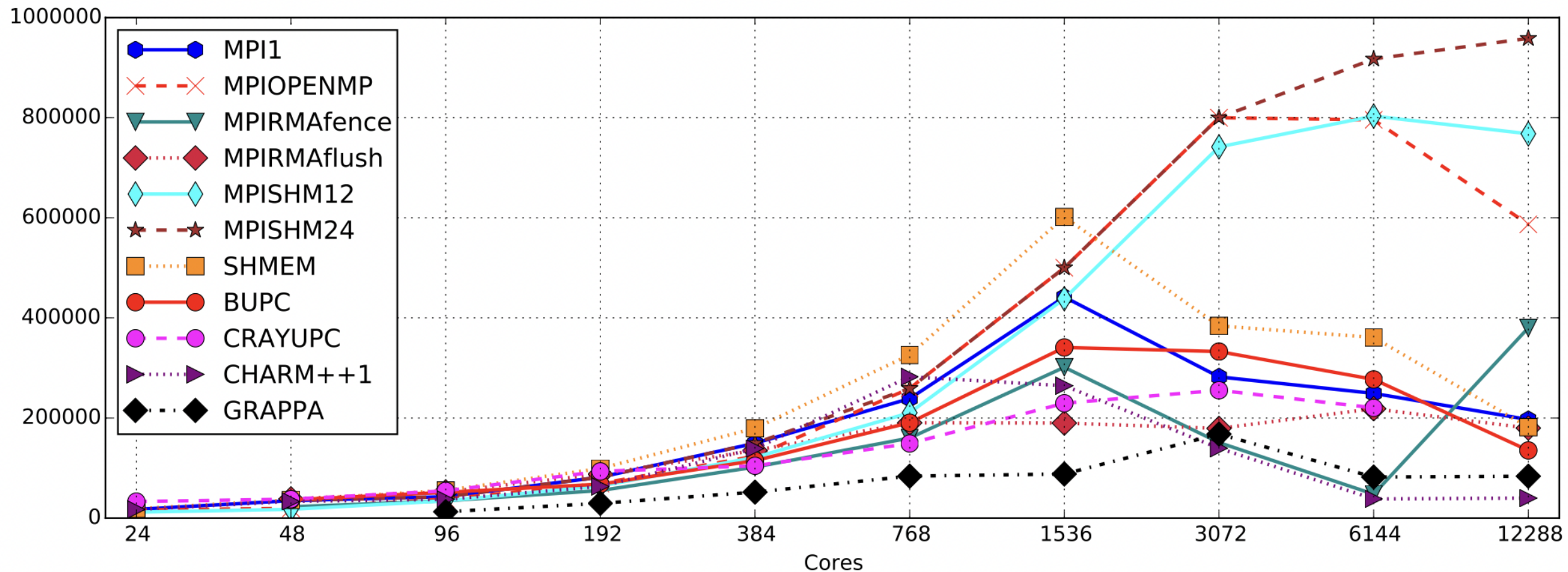
² University of Washington, Seattle, WA, USA.

<https://link.springer.com/book/10.1007/978-3-319-41321-1>

p2p on Cray XC30



transpose on Cray XC30



1536: 8 KiB/PE (flat)
 4.5 MiB/PE (hybrid)
 12288: 128 B/PE (flat)
 72 KiB/PE (hybrid)

MPI for Exascale! Now what?

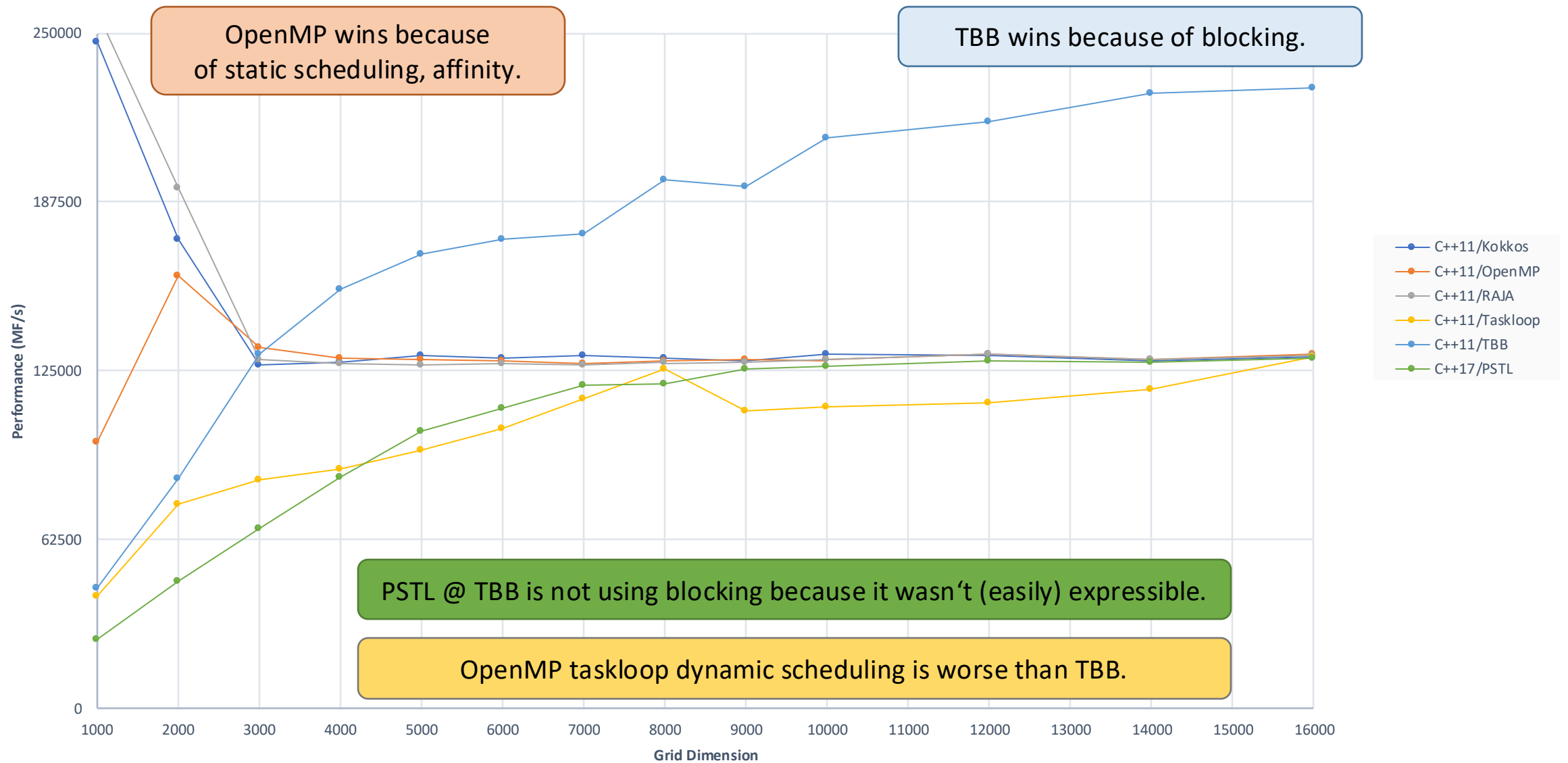
Something tells me that C89 and OpenMP 3.0 aren't enough...

Language	Seq.	OpenMP	MPI	PGAS	Threads	Others?
C89	√	√	√√√	SHMEM		
C11	√	√√√		UPC	√	Cilk, ISPC, PETSc
C++	√	√√√	RMA (WIP)	Grappa	√	Kokkos, RAJA, TBB, PSTL, SYCL, OpenCL, CUDA, HIP, ...
Fortran	√	√√√	√√√	Coarrays, GA		“pretty”, OpenACC
Python	√	√	√√√	SHMEM		Numpy/CuPy, Numba
Chapel	√		?	√		

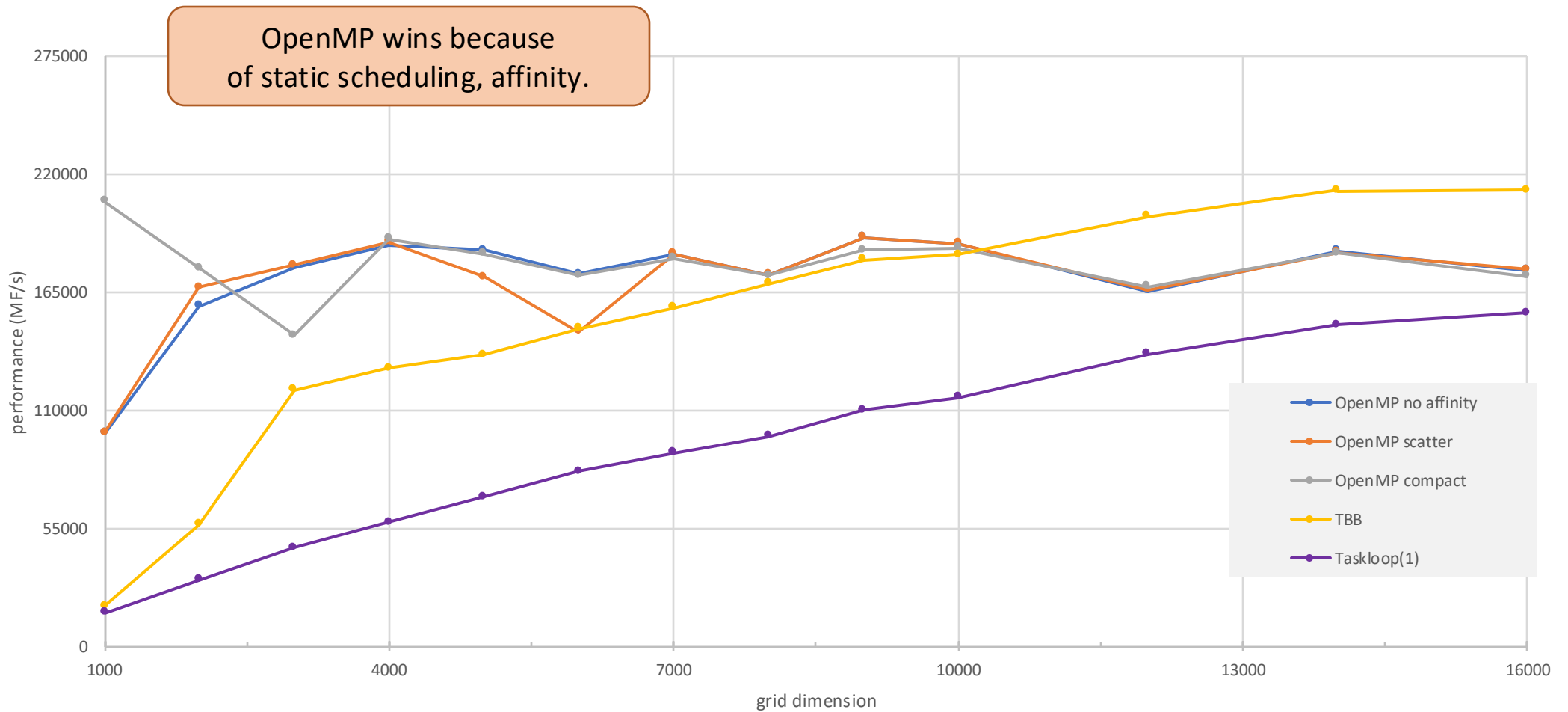
√√√ = OpenMP: traditional, task-based, and target are implemented similarly in Fortran, C and C++.

√√√ = MPI: two-sided, collective, one-sided (for transpose, at least).

PRK stencil: C++ implementations on KNL

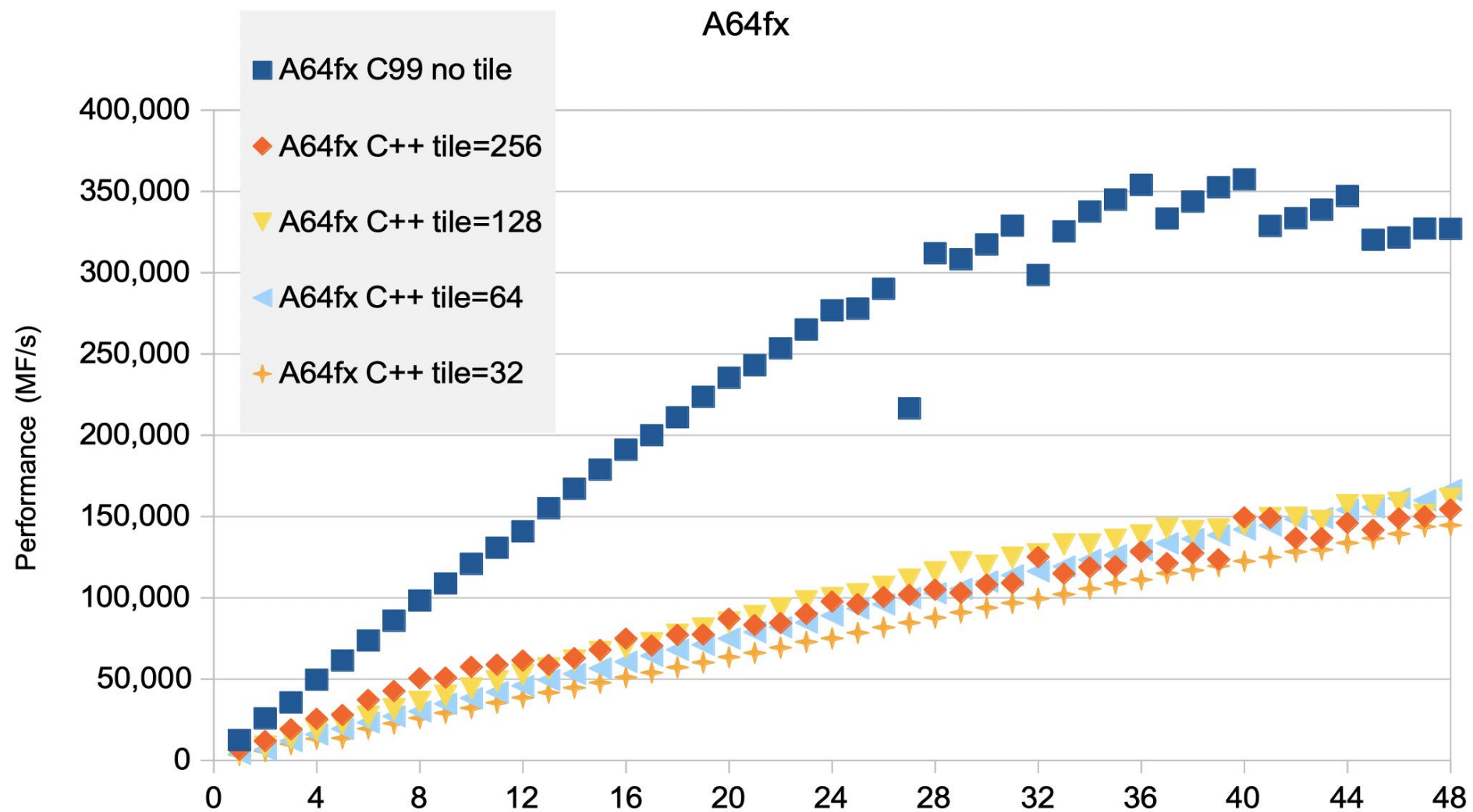


Improved PRK stencil on KNL



Fujitsu A64fx

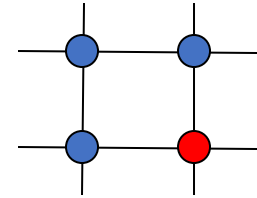
PRK stencil (2D, naive)



"This research was supported by the NSF MRI award #[1828187](#): "MRI: Acquisition of an HPC System for Data-Driven Discovery in Computational

Wavefront Parallelism

```
// sequential C implementation
for (int i=1; i<m; ++i) {
  for (int j=1; j<n; ++j) {
    A[i][j] = A[i-1][j] + A[i][j-1] - A[i-1][j-1];
  }
}
```

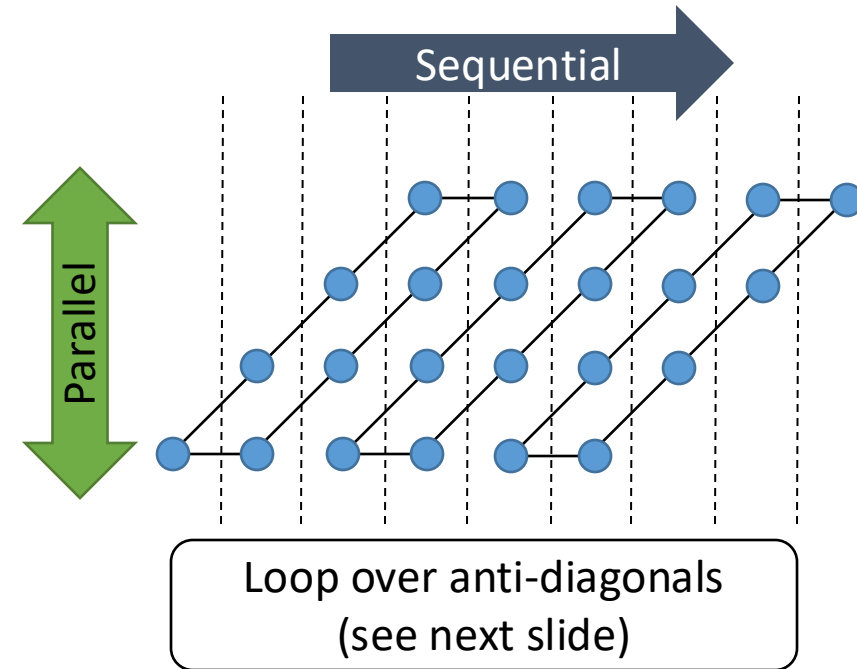
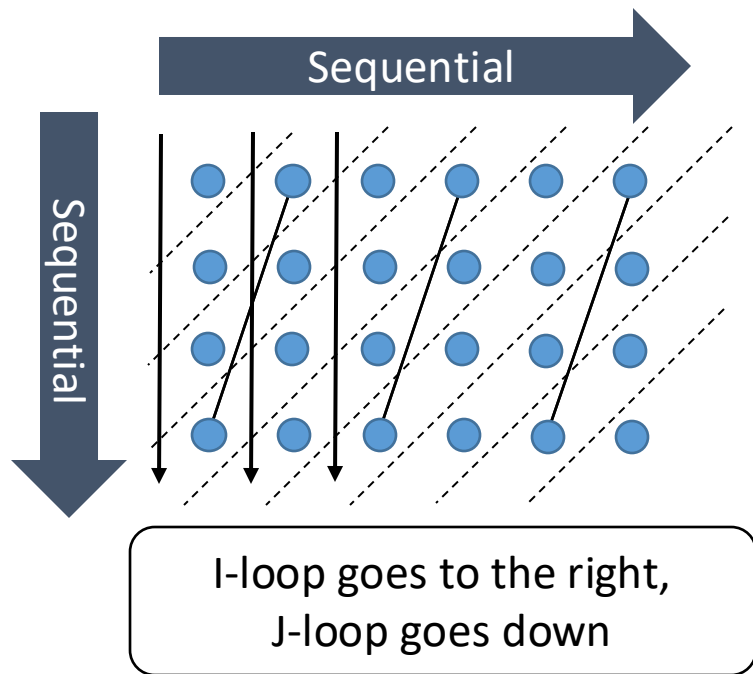


$$A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}$$

This pattern appears in a range of applications:

- Deterministic neutron transport (DOE-NNSA mission science)
- Smith-Waterman/PairHMM (bioinformatics), dynamic programming

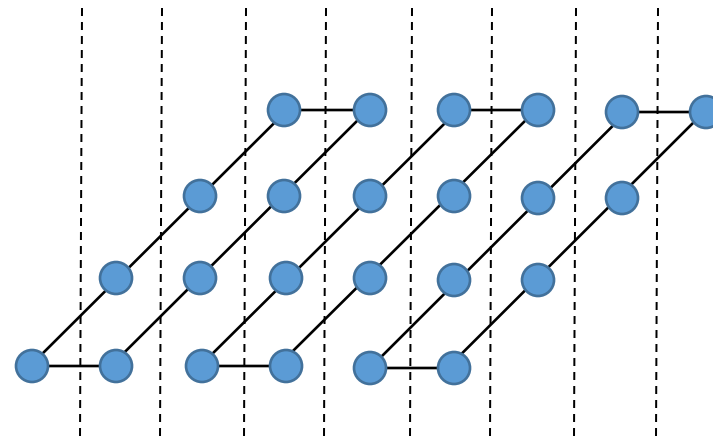
Changing the iteration space exposes parallelism



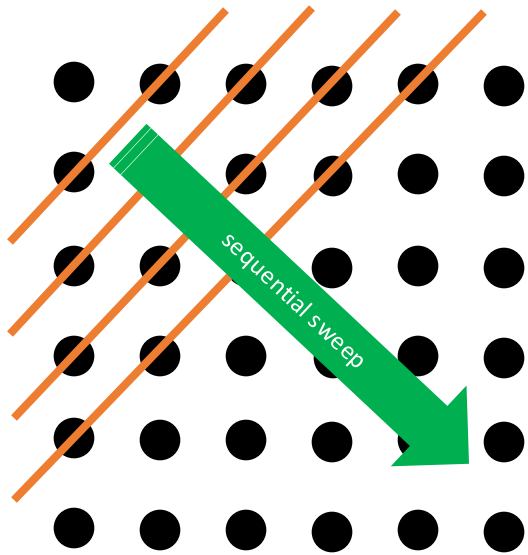
OpenMP inner-loop parallelism

```
// sequential loop
for (int i=2; i<=2*n-2; ++i) {
    int start = max(2,i-n+2);
    int stop  = min(i,n);
    #pragma omp for simd
    for (int j=start; j<=stop; ++j) {
        const int x = i-j+2-1;
        const int y = j-1;
        A[x][y] = A[x-1][y]
                + A[x][y-1]
                - A[x-1][y-1];
    }
    // implicit barrier (required)
}
```

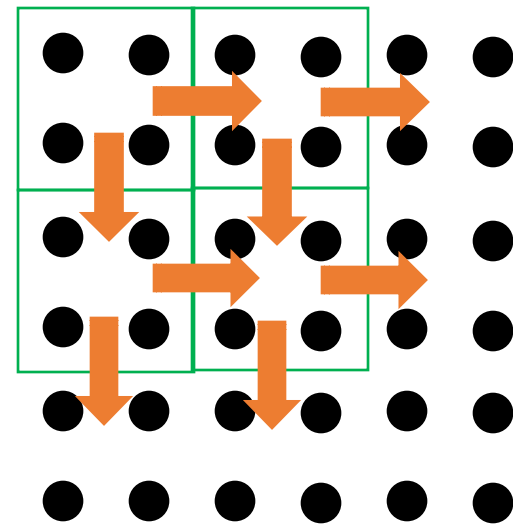
- Very low parallel efficiency once data spills private cache.
- CPU SIMD doesn't work because data access is non-contiguous.



Amortizing synchronization overheads



Parallel loop

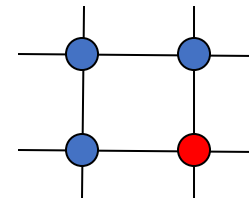


Task dependency

- Sequential execution requires no synchronization.
- Formally, there are $O(n^2)$ element-wise dependencies.
- Antidiagonal implementation uses $O(n)$ barriers to enforce deps.
- Hyperplane amortizes barriers across many antidiagonals: $O(n/\text{unroll})$ barriers.
- Task-based has $O(n^2/\text{block}^2)$ dependencies.

OpenMP task-based parallelism

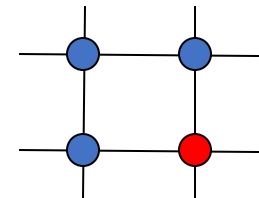
```
#pragma omp parallel
#pragma omp master
for (int i=1; i<m; i+=mc) {
  for (int j=1; j<n; j+=nc) {
    #pragma omp task depend(in:grid[i-mc][j],grid[i][j-nc]) \
                          depend(out:grid[i][j])
    for (int ii=i; ii<std::min(m,i+mc); ii++) {
      for (int jj=j; jj<std::min(n,j+nc); jj++) {
        A[ii][jj] = A[ii-1][jj] + A[ii][jj-1] - A[ii-1][jj-1];
      }
    }
  }
}
#pragma omp taskwait
```



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

OpenMP “doacross” parallelism

```
#pragma omp for collapse(2) ordered(2)
for (int i=0; i<ib; i++) {
  for (int j=0; j<jb; j++) {
    #pragma omp ordered depend(sink: i-1,j) depend(sink: i,j-1)
    for (int ii=i; ii<std::min(m,i+mc); ii++) {
      for (int jj=j; jj<std::min(n,j+nc); jj++) {
        A[ii][jj] = A[ii-1][jj] + A[ii][jj-1] - A[ii-1][jj-1];
      }
    }
    #pragma omp depend(source)
  }
}
```



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

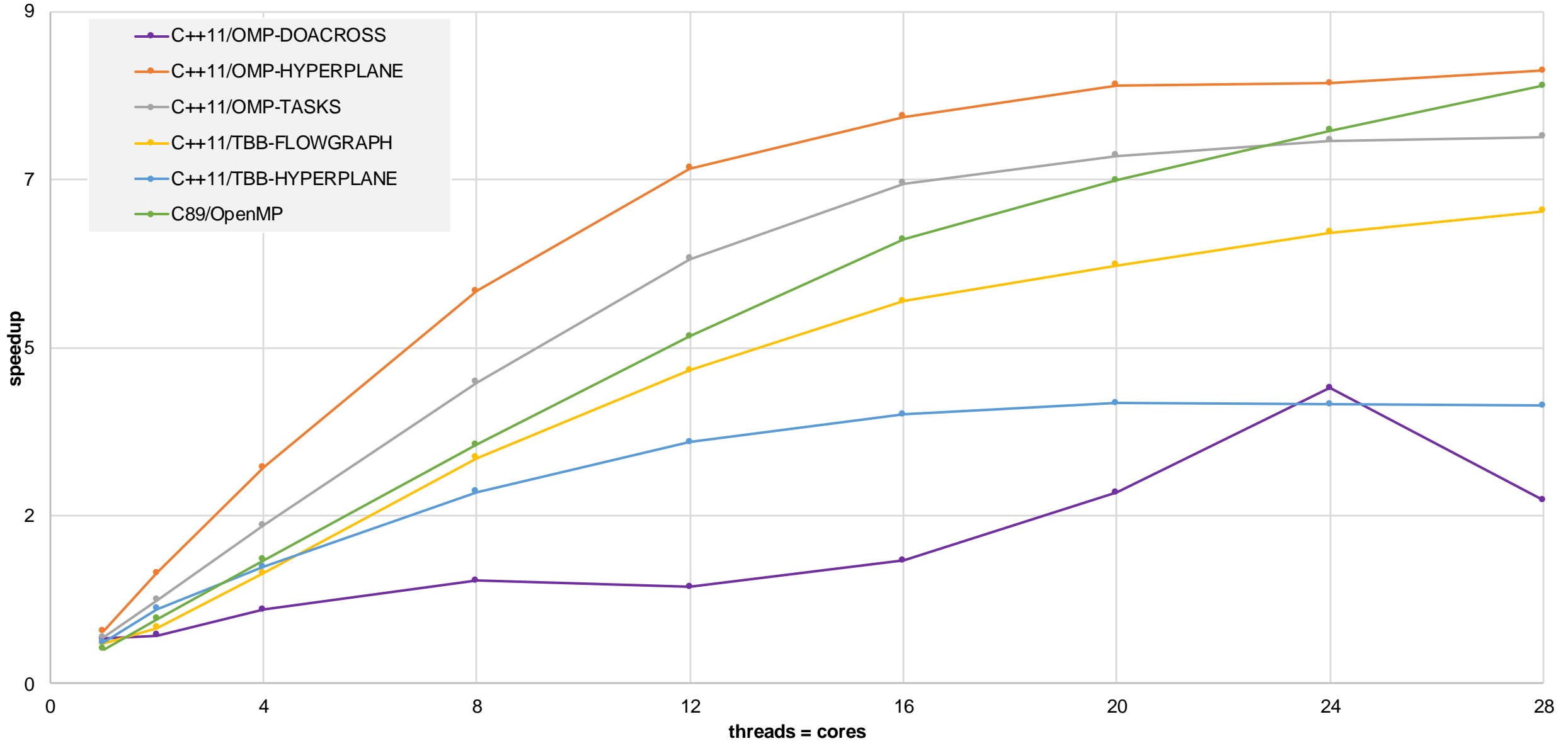
OpenMP hyperplane parallelism

```
#pragma omp parallel
for (int i=2; i<=2*(nb+1)-2; i++) {
    #pragma omp for
    for (int j=std::max(2,i-(nb+1)+2); j<=std::min(i,nb+1); j++) {
        const int ib = nc*(i-j)+1;
        const int jb = nc*(j-2)+1;
        for (int ii=ib; ii<std::min(m,ib+nc); ii++) {
            for (int jj=jb; jj<std::min(n,jb+nc); jj++) {
                A[ii][jj] = A[ii-1][jj] + A[ii][jj-1] - A[ii-1][jj-1];
            }
        }
    }
}
```

This is only implemented for square grids to keep the polyhedral arithmetic simpler.

Intel Skylake Xeon 8180 (1S)

PRK wavefront: grid=5000



Further analysis

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

2018:

https://github.com/ParRes/Kernels/blob/main/doc/IXPUG_Invited2_Hammond.pdf

2019:

<https://github.com/ParRes/Kernels/blob/main/doc/Hammond-PPP2019.pdf>

<https://dl.acm.org/doi/10.1145/3318170.3318192>

Shifting through the Gears of GPU Programming: Understanding Performance and Portability Trade-offs

2022:

<https://www.nvidia.com/en-us/on-demand/session/gtcspring22-s41620/>

What if we used the same methods to evaluate more programming languages?

Language	Nstream	Transpose	Stencil	Synch_p2p	Dgemm
Ada	√	√			
C#	√	√			
Go	√	√			√
Java	√	√	√	√	
Julia	√√	√√	√	√	√√
Lua	√				
Octave (Matlab)	√√	√√	√√	√	
Python	√√√	√√√	√√√	√	√√o
Ruby	√				
Rust	√√	√	√	√	√√
Scala	√				

√√ = Traditional (loops) and Pretty (higher-level)

√√√ = Traditional (loops), Numpy, MPI (mpi4py)

Do not assume that Jeff knew what he was doing when he wrote Go, Lua, Rust, ... well anything frankly. Outside contributions are more reliable.

Nstream language showdown

Language	MB/s
C89	31,906
Fortran	33,221
C11	34,529
C++11	33,460
Rust	21,576
Java	32,264
Go	24,546
Julia loops	34,193
Julia pretty	34,645
Numpy	17,229
Octave colon	9,804
<i>Lua</i>	273
<i>Python</i>	262
<i>Octave loops</i>	4

Language	MB/s
C89 (OpenMP)	43,977
Fortran pretty	42,399
Fortran loops	42,186
Fortran (GPU)	959,310
C11	40,540
C++	39,356
C++ vector	41,801
C++ valarray	41,849
C++ range-for	40,301
C++ for_each (GPU)	957,674
Ada	8,912
Rust	37,566
Rust unsafe	39,265
Rust iter	38,907
Rust rayon	43,334
Java	35,753
Go	24,882
Julia loops	40,037
Julia pretty	39,098
Numpy	17,003
CuPy (GPU)	469,728
Numba	41,159

Left: N=1Mi, GCC 9, Ubuntu 20.04, Tiger Lake CPU

Right: N=128Mi, NVHPC 24.9, Ubuntu 24.04, Zen4 CPU

GPU memory systems

GPU Nstream

		A100 40G	A100 80G	H100 SMX5	GH200 480GB	MI-100	MI-210	MI-250x 1 GCD only	PVC 1110 1 tile SKU
Base Language	Programming Environment	NVHPC	NVHPC	NVHPC	NVHPC	ROCM	ROCM	Cray PE or HIP	oneAPI
C++	OpenCL	1,359,420	1,797,890	3,130,630	3,758,870	976,608	1,178,010	N/A	785,072
C++	CUDA / HIP / DPC++	1,376,040	1,806,090	3,137,810	3,786,340	985,792	1,247,460	1,271,440	786,219
C++	CUDA / HIP / DPC++ MM / USM	1,352,760	1,793,550	3,127,140	3,745,890	34,477	31,015	28,095 1,282,240 (xnack 1Gi)	788,243
C++	StdPar / oneDPL		1,783,730	3,122,440	3,735,440	-	-	-	786,460
C++	OpenMP target	1,331,240	1,696,960	3,011,500	3,736,180	813,976	1,032,640	1,279,560	788,934
Fortran	OpenMP target	1,318,457	1,695,184	3,123,503	3,624,402	750,497	748,734	1,211,633	674,388
Fortran	OpenACC	1,396,610	1,775,602	3,137,305	3,713,856	N/A	N/A	1,215,681	N/A
Fortran	StdPar		1,625,644	3,122,048	3,627,525	N/A	N/A	N/A	11,049

See also: <https://github.com/UoB-HPC/BabelStream/>

NWChem Co-Design Insight w/ PRK DGEMM

Matrix Dimension	Compute Time (ms)	Transfer Time (ms)	Compute Rate (TF/s) post-transfer
5000	4.0	29.6	62.0
10000	31.4	36.1	63.6
20000	248.5	27.9	64.5

DGX-H100 running PRK DGEMM with managed memory.
Transfer time is the compute time of the first iteration minus the second iteration; there is some overlap between GPU page faults and compute.

Other notable activities of late...

- CUDASTF single- and multi-GPU evaluation.
- Julia and Rust updates, including parallel dialects.
- Fortran MPI, mpi4py, shmem4py
- NCCL and NVSHMEM (just started)
- <your amazing contribution here>



Summary

- The PRK project is a research project for understanding:
 - Communication semantics and implementation details, e.g. MPI vs SHMEM
 - Toolchain implementation quality, e.g. GCC vs LLVM
 - Programming model semantics and syntax, e.g. SYCL vs Kokkos
 - Programming language differences, e.g. Fortran vs Julia, C++ vs Python
- The PRK project is not particularly useful for understanding absolute performance or hardware characteristics *unless you tune the code appropriately.*
- PRK kernels run in cycle-accurate chip simulators, Raspberry Pi, all the GPUs, and large-scale supercomputers. You can use them to co-design anything!

main [11 Branches](#) [6 Tags](#)

Go to file

[Code](#)








 jeffhammond Hammond IXPUG April 2018 KAUST - PRK C++ ...	a55b710 · 10 hours ago	 3,772 Commits
.github	add GitHub Actions, mostly remove Travis CI (#593)	3 years ago
ADA	transpose work in progress	9 months ago
AMPI	avoid overflow	5 years ago
C1z	cleanup	3 weeks ago
CHARM++	avoid overflow	5 years ago
Csharp	make input parsing work with older C# compilers	4 years ago
Cxx11	restore macro that was used	10 hours ago
FENIX	Finalizing Fenix Transpose kernel.	7 years ago
FG_MPI	remove C99 flag (#454)	4 years ago
FORTRAN	cosmetic changes	last year
GO	add docs for Go that work now	3 weeks ago
GRAPPA	Fixing makefile comment about default shape for stencils.	7 years ago
JAVA	make may not be the right thing for Java but it is okay for ...	4 years ago
JULIA	not sure if this is indeed better	yesterday
LEGION	Fix for mapper interface update.	5 years ago
LUA	add Lua nstream	4 years ago
MPI1	add SHMEM C transpose with alltoall	last year
MPIOPENMP	apply the same fix to MPI+OpenMP	9 months ago
MPIRMA	remove C99 flag (#454)	4 years ago
MPISHM	remove C99 flag (#454)	4 years ago

About

This is a set of simple programs that can be used to explore the features of a parallel platform.

[groups.google.com/forum/#!forum/par...](#)

[c](#) [c-plus-plus](#) [hpc](#) [julia](#) [opencl](#)
[parallel](#) [openmp](#) [mpi](#) [python3](#)
[pgas](#) [coarray-fortran](#) [threading](#)
[openacc](#) [kokkos](#) [shmem](#) [sycl](#)
[parallel-programming](#) [fortran2008](#)

-  Readme
 -  View license
 -  Activity
 -  Custom properties
 -  414 stars
 -  39 watching
 -  109 forks
- [Report repository](#)

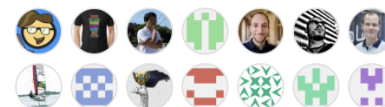
Releases

 6 tags

Packages

No packages published

Contributors 31



[+ 17 contributors](#)

<https://github.com/ParRes/Kernels>

The End