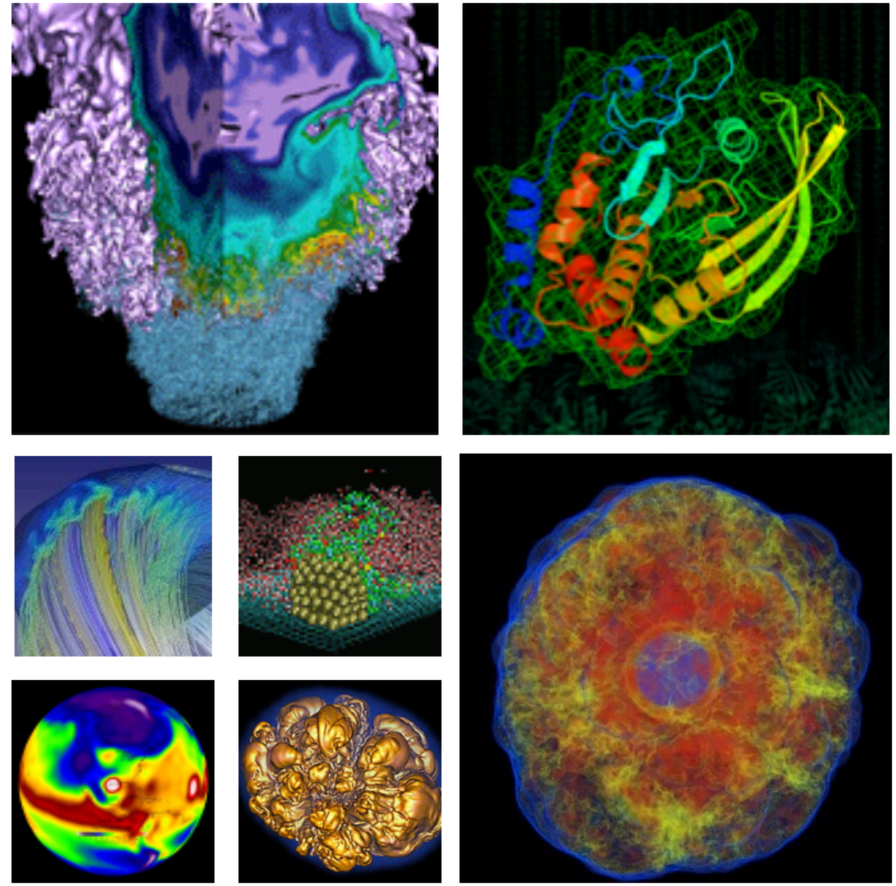
Guide for applying to Cori P2 early access (KNLEAP)



Thorsten Kurth

Cori KNL Training
November 3, 2016

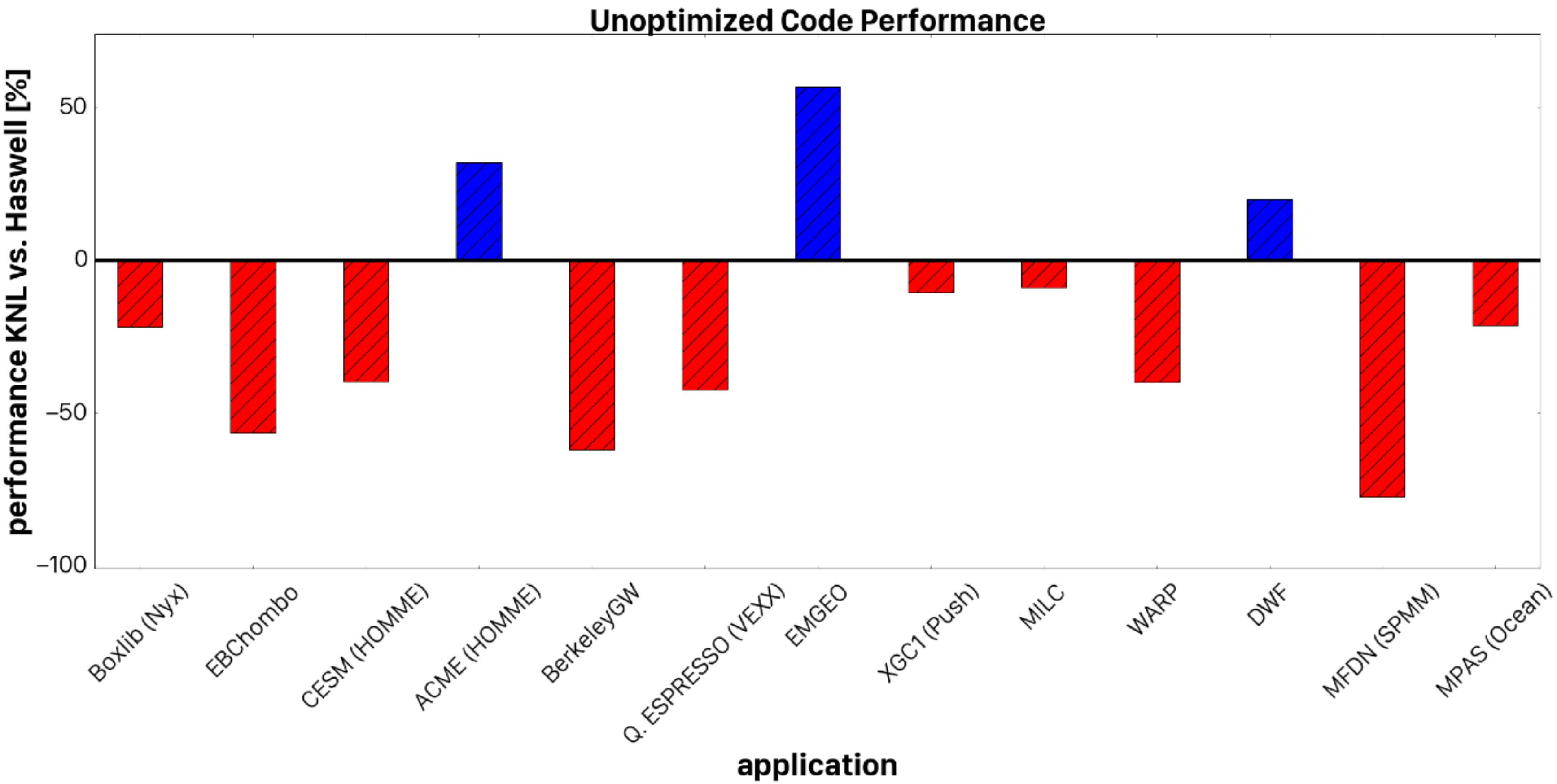
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Do we need to do this?

- KNL **is not** different
  - x86-64 compatible
  - self-hosted
  - compile and run
- KNL **is** different
  - slow sequential cores (≈1.2Ghz vs. ≈2.3Ghz)
  - high-bandwidth on-package memory but no L3
  - lots of cores (68/272 vs. 32/64)
  - wide vector units (512bit vs. 256bit)

Why are we doing this?



- don't expect good performance out of the box

Why are we doing this?

- make users think about their code
  - why am I getting the performance I am seeing?
  - can I do better?
  - how can I do better?
- gather data to inform future NERSC procurements
  - what kinds of codes are running at NERSC
  - what architectures might be most beneficial to users?
- make Cori Phase II a success for everybody

Overview

- General Section
  - repository name and users which should be enabled, application/code name, science description, programming languages, etc.
- Performance Section
  - guided set of experiments on Haswell and KNL partition
  - show us that your code is ready
- visit https://my.nersc.gov/knleap.php and log in

General Section

- **Repository name**: you apply on behalf of users in that repository
- **Repository user list**: select up to 5 users to grant access if applications is accepted
- **Application Code**: the name of your code
- **Science Description**: please describe briefly what your science plans are and what partition sizes you want to use
- **Programming languages**: tell us what your code is made of
- **Application Kernels**: what are the hotspots/workhorses in your application? This helps us to give you hints for optimization

- **Thread scaling**
  - make familiar with hybrid MPI+OpenMP programming model
  - show how well your code utilizes threads
  - might help your code to scale-out
  - if your code uses MPI-only, might still be OK

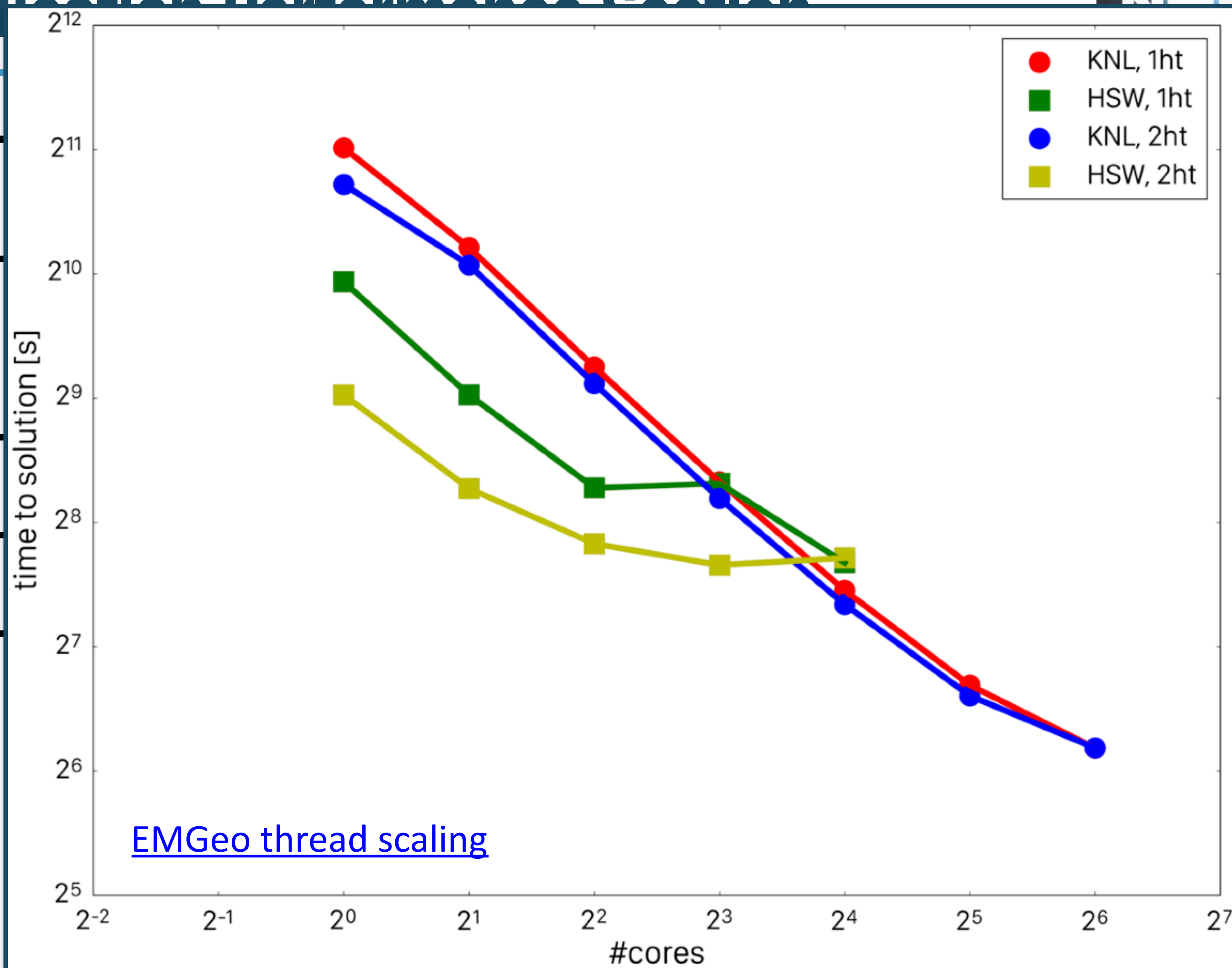
Application Performance Section

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -C haswell

#this is the Haswell script

echo ncores,nht,arch,time > threadscale_hsw.csv
for nc in 1 2 4 8 16 32; do
    for nht in 1 2; do
        export OMP_NUM_THREADS=$(( ${nc} * ${nht} ))
        export OMP_PLACES=cores“(${nc})“
        export OMP_PROC_BIND=spread

        srun -n 1 -c 64 --cpu_bind=cores ./my_hsw_app.x >
        output

        timing=<extract timing from output>

        echo ${nt},${nht},hsw,${timing} >> threadscale_hsw.csv
    done
done
```

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -C knl,quad,cache

#this is the KNL script

echo ncores,nht,arch,time > threadscale_knl.csv
for nc in 1 2 4 8 16 32 64; do
    for nht in 1 2 4; do
        export OMP_NUM_THREADS=$(( ${nc} * ${nht} ))
        export OMP_PLACES=cores“(${nc})“
        export OMP_PROC_BIND=spread

        srun -n 1 -c 272 --cpu_bind=cores ./my_knl_app.x >
        output

        timing=<extract timing from output>

        echo ${nt},${nht},knl,${timing} >> threadscale_knl.csv
    done
done
```

Appl

• T



KNL, 1ht
HSW, 1ht
KNL, 2ht
HSW, 2ht
time to solution [s]
#cores
EMGeo thread scaling

Application Performance Section

- **MPI vs. threading performance**
  - what is the #ranks/#thread sweet-spot for my application on a single node
  - might change when scaling out, but gives a good impression of what might be a reasonable setting
  - please keep NUMA effects in mind

Application Performance Section

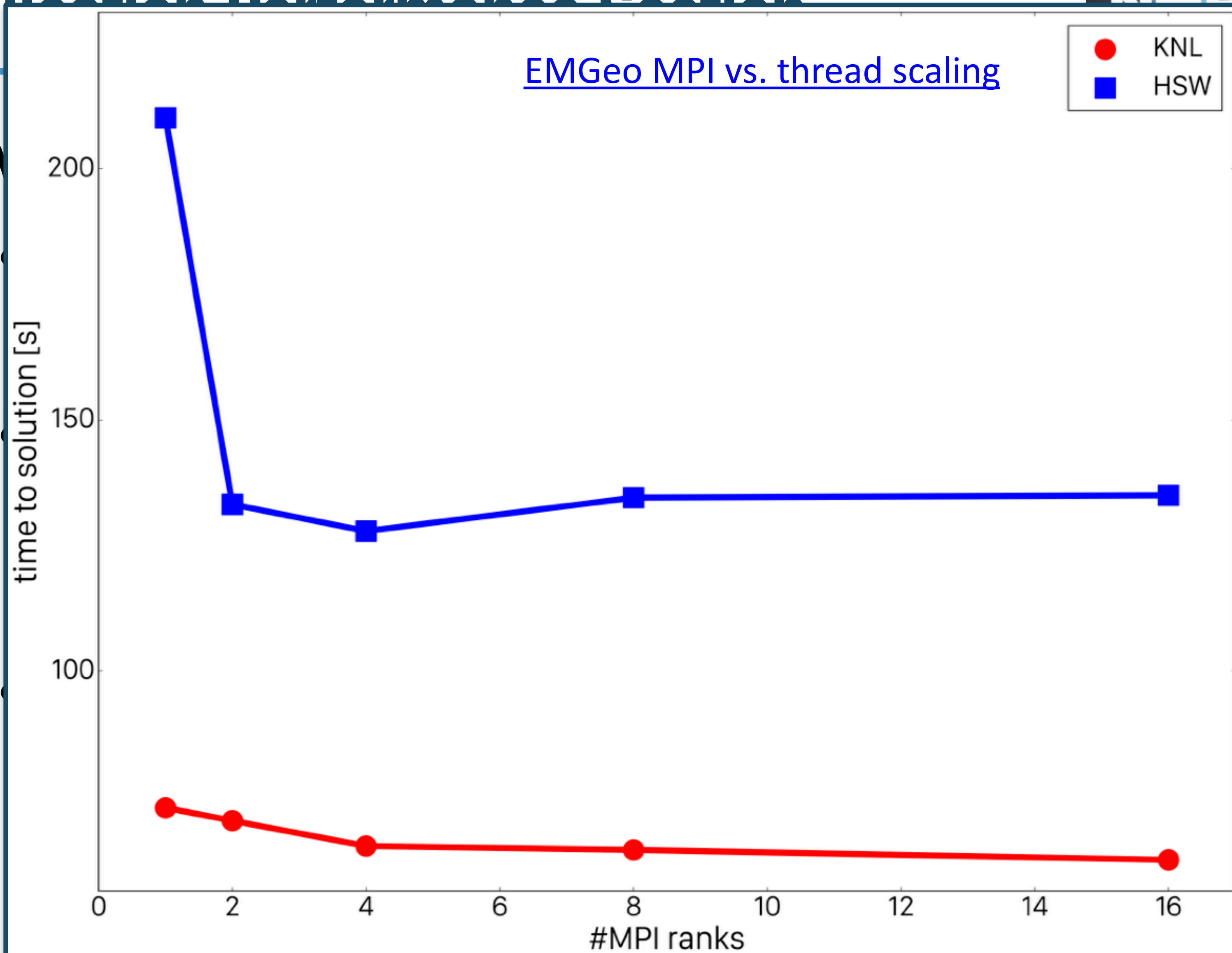
•

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -C haswell

#this is the Haswell script
export nht=<optimal number of hyperthreads>

echo nranks,arch,time > mpi_vs_threadscale_hsw.csv
for nr in 1 2 4 8 16 32; do

    export OMP_NUM_THREADS=$(( $(( 32 * ${nht} )) / ${nr} ))
    export OMP_PLACES=cores“($(( 32 / ${nr} )))“
    export OMP_PROC_BIND=spread

    srun -n 1 -c $(( 64 / ${nr} )) --cpu_bind=cores
                ./my_hsw_app.x input_${nr} > output

    timing=<extract timing from output>

    echo ${nr},knl,${timing} >> mpi_vs_threadscale_hsw.csv
done
```

Application Performance Section

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -C knl,quad,cache

#this is the KNL script
export nht=<optimal number of hyperthreads>

echo nranks,arch,time > mpi_vs_threadscale_knl.csv
for nr in 1 2 4 8 16 32 64; do

    export OMP_NUM_THREADS=$(( $(( 64 * ${nht} )) / ${nr} ))
    export OMP_PLACES=cores“($(( 64 / ${nr} )))“
    export OMP_PROC_BIND=spread

    srun -n 1 -c $(( 256 / ${nr} )) --cpu_bind=cores
                 ./my_knl_app.x input_${nr} > output

    timing=<extract timing from output>

    echo ${nr},knl,${timing} >> mpi_vs_threadscale_knl.csv
done
```

Application Performance Section



Application Performance Section

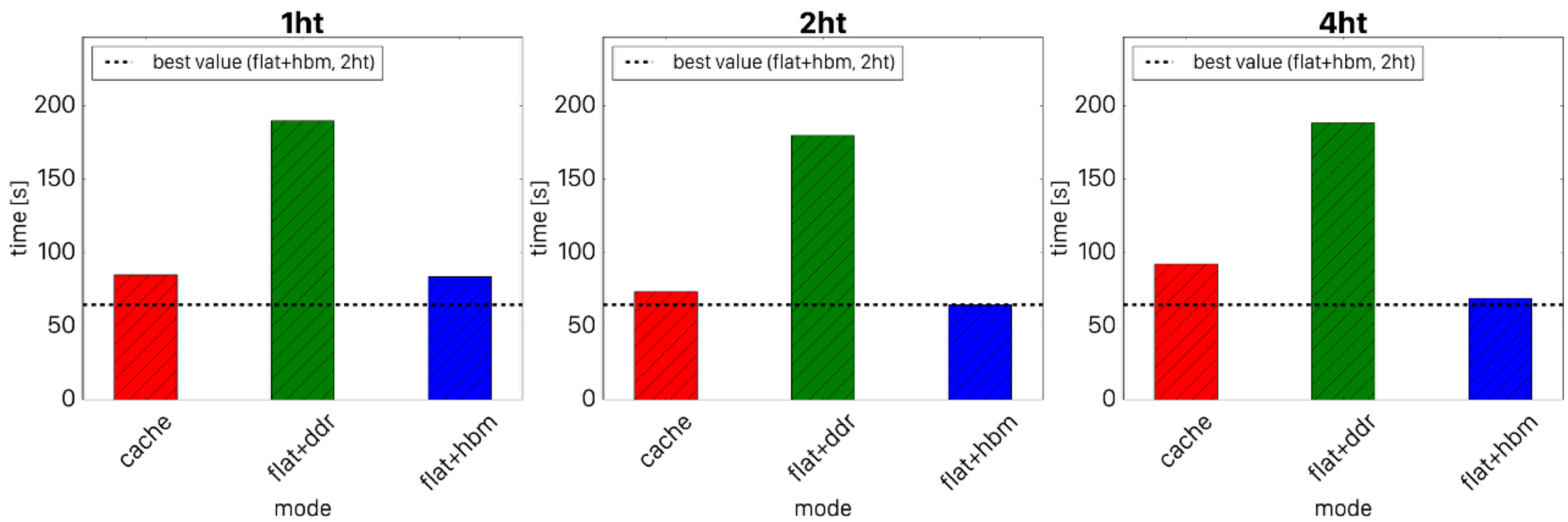
- **(Memory) mode comparison**
  - KNL chip and memory configuration can be changed
    - chip: Quadrant, SNC-2, SNC-4
    - memory: flat, cache, (hybrid)
  - make yourself familiar with how to use these modes since they can significantly affect the performance
  - experience: memory modes have bigger impact, we only ask you to test those (feel free to try other cpu modes as well)

Application Performance Section

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -C knl,quad,cache

#this is the KNL script
export nr=<optimal number of ranks>
export nc=<optimal number of cores>
export nht=<optimal number of hyperthreads>

#run
export OMP_NUM_THREADS=$(( $(( 64 * ${nht} )) / ${nr} ))
export OMP_PLACES=cores“($(( 64 / ${nr} )))“
export OMP_PROC_BIND=spread

echo mode,time > mode_comparison_knl.csv
srun -n 1 -c $(( 256 / ${nr} )) --cpu_bind=cores
             ./my_knl_app.x input_${nr} > output

timing=<extract timing from output>

echo cache,${timing} >> mode_comparison_knl.csv
```

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -C knl,quad,flat

#this is the KNL script
export nr=<optimal number of ranks>
export nc=<optimal number of cores>
export nht=<optimal number of hyperthreads>

#run
export OMP_NUM_THREADS=$(( $(( 64 * ${nht} )) / ${nr} ))
export OMP_PLACES=cores“($(( 64 / ${nr} )))“
export OMP_PROC_BIND=spread

#flat+ddr
srun -n 1 -c $(( 256 / ${nr} )) --cpu_bind=cores
             numactl -p 0 ./my_knl_app.x input_${nr} > output
timing=<extract timing from output>
echo „flat+ddr“,${timing} >> mode_comparison_knl.csv

#flat+hbm
srun -n 1 -c $(( 256 / ${nr} )) --cpu_bind=cores
             numactl -p 1 ./my_knl_app.x input_${nr} > output
timing=<extract timing from output>
echo „flat+hbm“,${timing} >> mode_comparison_knl.csv
```

Application Performance Section

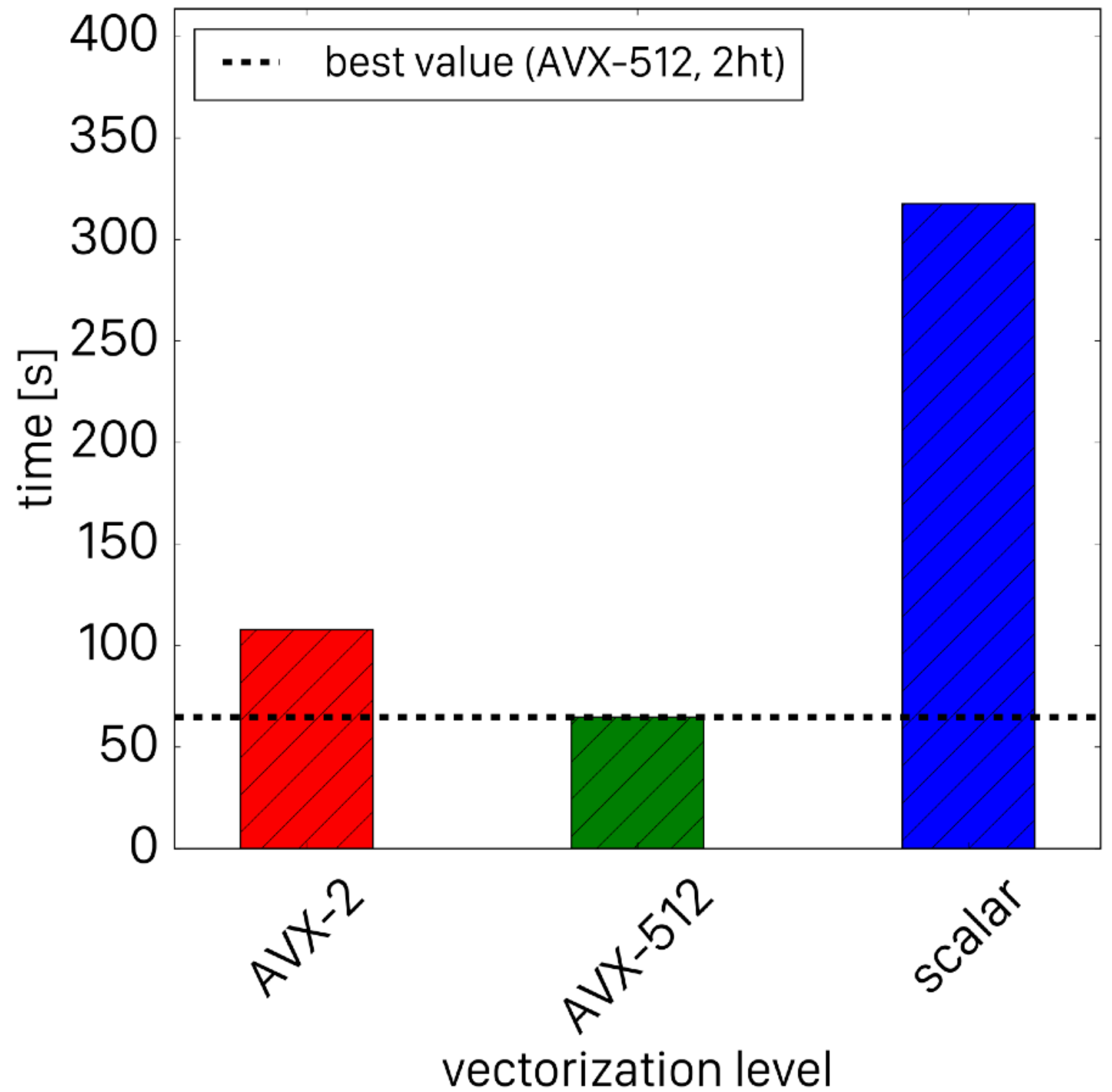


EMGeo mode comparison

Application Performance Section

- **Vectorization experiment**
  - does your code benefit from larger vector width in KNL?
  - is your code vectorized efficiently?
- remark: if the main workload in your code comes from external library calls, then you might not see a big difference in this experiment

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p regular
#SBATCH -C knl,quad,<optimal mode>

#this is the KNL script
export nr=<optimal number of ranks>
export nc=<optimal number of cores>
export nht=<optimal number of hyperthreads>

#run
export OMP_NUM_THREADS=$(( $(( 64 * ${nht} )) / ${nr} ))
export OMP_PLACES=cores“($(( 64 / ${nr} )))“
export OMP_PROC_BIND=spread

echo mode,time > vec_comparison.csv

#scalar
srun -n 1 -c $(( 256 / ${nr} )) --cpu_bind=cores <mode-dep-prefix> ./
my_scalar_knl_app.x input_${nr} > output
timing=<extract timing from output>
echo scalar,${timing} >> vec_comparison_knl.csv

#avx2
srun -n 1 -c $(( 256 / ${nr} )) --cpu_bind=cores <mode-dep-prefix> ./
my_avx2_knl_app.x input_${nr} > output
timing=<extract timing from output>
echo "AVX-2",${timing} >> vec_comparison_knl.csv

#avx512
srun -n 1 -c $(( 256 / ${nr} )) --cpu_bind=cores <mode-dep-prefix> ./
my_knl_app.x input_${nr} > output
timing=<extract timing from output>
echo "AVX-512",${timing} >> vec_comparison_knl.csv
```



[EMGeo vectorization study]

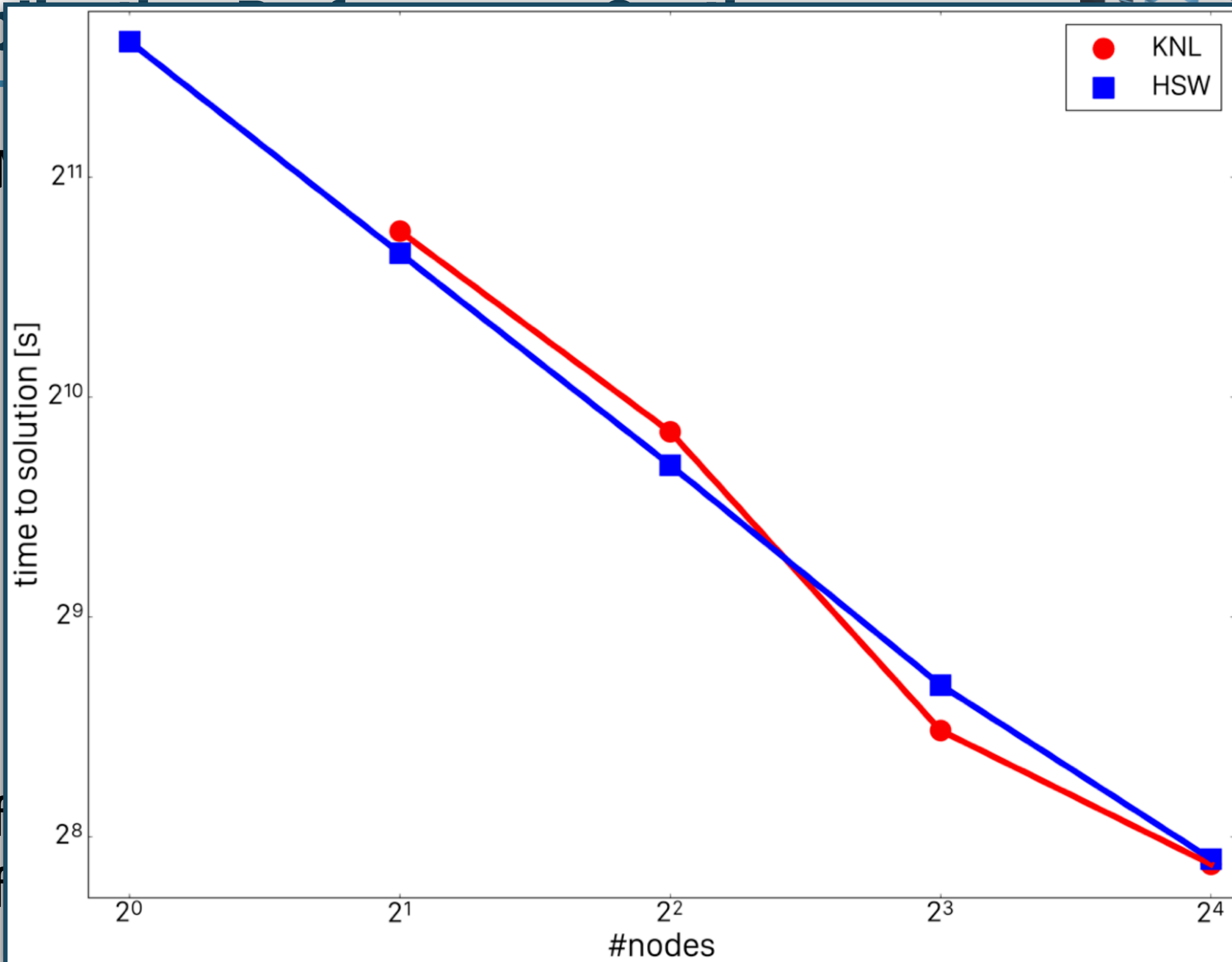
Application

	no-vectorization	AVX-2	AVX-512
Intel	-no-vec -no-simd (comment out SIMD)	-xCORE-AVX2	-xMIC-AVX512
GNU	-march=knl -fno-tree-vectorize -fno-tree-loop-vectorize -fno-tree-slp-vectorize	-march=knl -mavx2	-march=knl
Cray	-h vector0 -h nopattern	-h cpu=haswell	-h cpu=mic-knl

- overview over compilation flags for this experiment

Application Performance Section

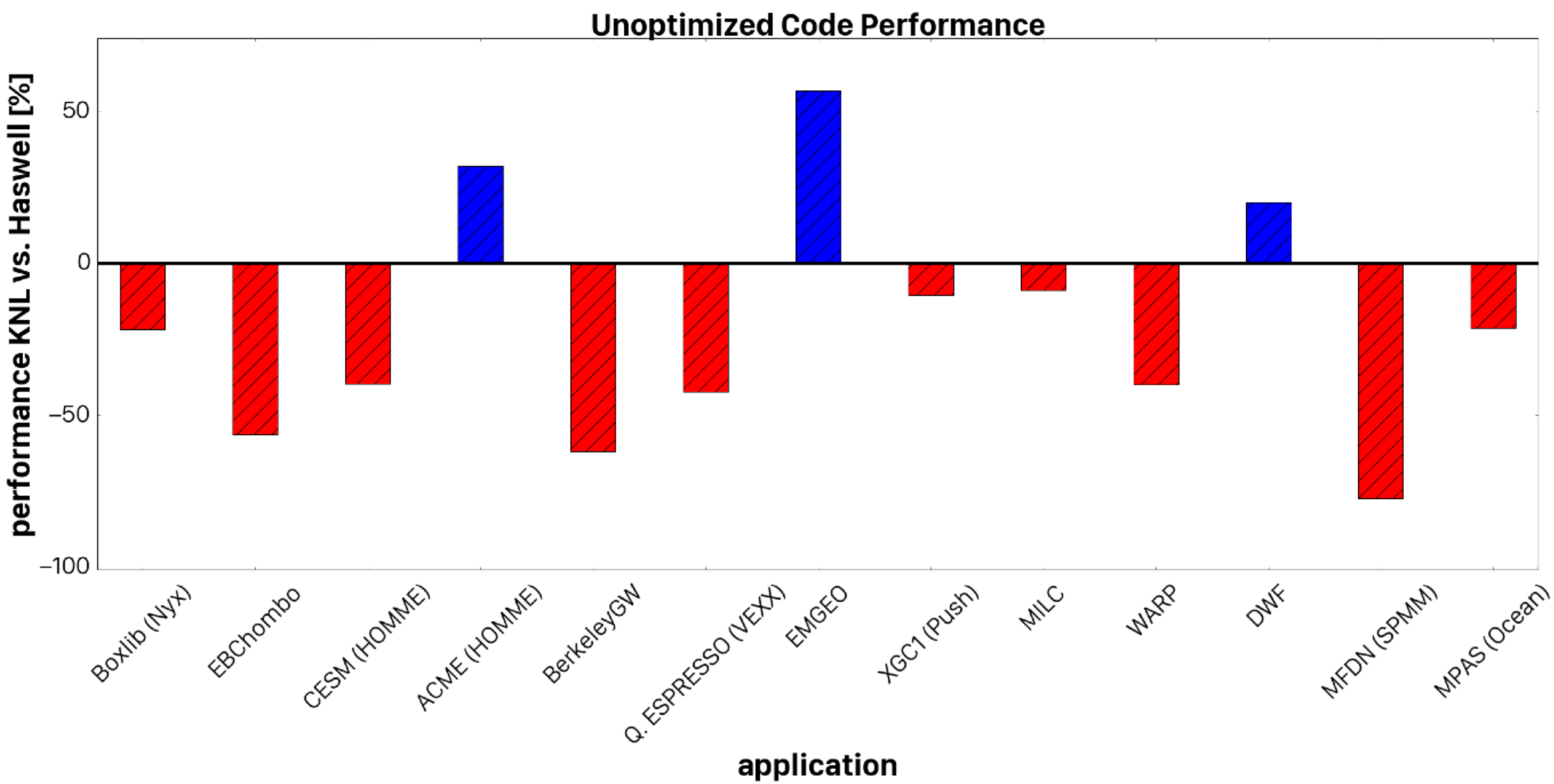
- **Multi-node scaling**
  - show us how good your code scales
  - demonstrate scaling according to character of production runs
    - capability: strong scaling (fixed global problem size)
    - capacity: weak scaling (fixed local problem size)
    - or both
- find the optimal choices for mode/#ranks_per_node for each number of nodes


KNL
HSW
time to solution [s]
#nodes
2^{8}
2^{9}
2^{10}
2^{11}
2^{0}
2^{1}
2^{2}
2^{3}
2^{4}

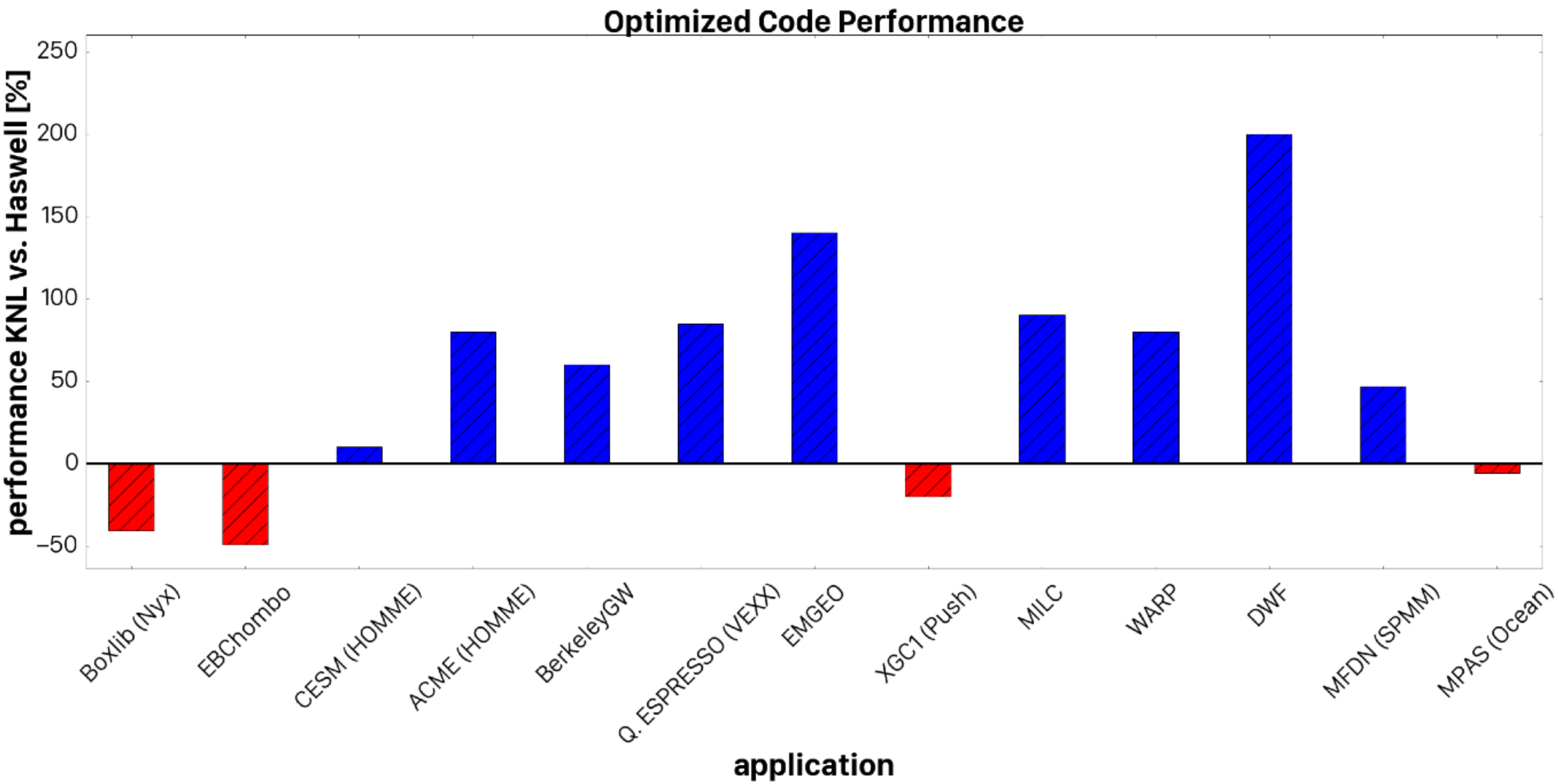
Final Remarks

- visit https://my.nersc.gov/knleap.php and log in
- fill out the form and submit
- note that once someone submitted an application for a given repo, **he will become the point of contact.**
- **please save your application data manually**
- applications can be retrieved as PDF on request, send email to consult@nersc.gov
- if rejected: see what you can do in the upcoming talks about profiling and review our case studies

What can be achieved?



What can be achieved?



NERSC

Thank you