



# *LIBXSMM*

*Library for small matrix multiplications.*

*Intel High Performance and Throughput Computing (EMEA)*

*Hans Pabst, July 7<sup>th</sup> 2015*



# Abstract

*Library for small matrix-matrix multiplications targeting Intel Architecture (x86). The library generates code for the following instruction set extensions: Intel SSE3, Intel AVX, Intel AVX2, IMCI (KNCni) for Intel Xeon Phi coprocessors ("KNC"), and Intel AVX-512 as found in the Intel Xeon Phi processor family ("KNL") and future Intel Xeon processors.*

*Historically the library was solely targeting the Intel Many Integrated Core Architecture "MIC") using intrinsic functions, however meanwhile optimized assembly code is generated for the fore mentioned instruction set extensions.*

# Motivation

## *“Improving Performance for Small Size Problems\*.”*

### **Make informed tradeoffs and gain performance**

- Generating specialized code “everything is hard/hand-coded”
- Highly optimized code “assembly, Intrinsics, and tuned code”

\* Actual problem size might be large when processing batches of small-sized problems.

# *Intel Math Kernel Library (Intel MKL)*

**DIRECT CALL feature:** Intel MKL allows inlining code for very small problem sizes as well as calling the low-level library implementation directly for small problem sizes. This feature may improve performance because of skipping error checks and calls to intermediate library layers.

- Works for Intel and non-Intel compilers
- Works for C/C++ and Fortran
- Compile-time decision

Improved performance for small problem sizes.

# Intel MKL 11.2: DIRECT CALL

## *Tradeoffs and Limitations*

BLAS-conformant error checking vs. low overhead

- No error checking or 'xerbla' callback

Code dispatch vs. compile-time decision

- AVX, AVX2, no MIC code path

Subset of functions LAPACK/BLAS3

- xGEMM

**Make informed tradeoffs and gain performance.**

# Intel MKL 11.2: VERBOSE Mode

***Quickly check if an application performs small matrix multiplications, and estimate the performance impact.***

1. Set the environment variable MKL\_VERBOSE=1
2. Select a representative workload
3. Run with redirected standard output

Example: evaluate occurrences of DGEMM for M, N, and K

```
$ env MKL_VERBOSE=1 ./myapplication > verbose.txt
```

```
$ grep -a "MKL_VERBOSE DGEMM" verbose.txt | cut -d, -f3-5
```

# *LIBXSMM*

**Library for small matrix multiplications.**

# LIBXSMM

## Interface (C API)

Simplified interface for matrix-matrix multiplications

- $c_{m \times n} = c_{m \times n} + a_{m \times k} * b_{k \times n}$  (no full xGEMM)

Dispatched and non-dispatched code paths

- Specialized/generated, or inlined C code
- LAPACK/BLAS (fallback code path)
- Amortized dispatch (“zero” overhead)

## License

- Open Source Software (BSD 3-clause license)\*

\* <https://github.com/hfp/libxsmm>



# LIBXSMM: Interface (C API)

```
/** If non-zero function pointer is returned, call (*function)(M, N, K). */
libxsmm_smm_function libxsmm_smm_dispatch(int m, int n, int k);
libxsmm_dmm_function libxsmm_dmm_dispatch(int m, int n, int k);

/** Automatically dispatched matrix-matrix multiplication. */
void libxsmm_smm(int m, int n, int k,
                 const float* a, const float* b,
                 float* c);
void libxsmm_dmm(int m, int n, int k,
                 const double* a, const double* b,
                 double* c);

/** Non-dispatched matrix-matrix multiplication using inline code. */
void libxsmm_simm(int m, int n, int k,
                 const float* a, const float* b,
                 float* c);
void libxsmm_dimm(int m, int n, int k,
                 const double* a, const double* b,
                 double* c);

/** Matrix-matrix multiplication using BLAS. */
void libxsmm_sblasmm(int m, int n, int k,
                    const float* a, const float* b,
                    float* c);
void libxsmm_dblasmm(int m, int n, int k,
                    const double* a, const double* b,
                    double* c);
```

# LIBXSMM: Getting Started

```
#include <libxsmm.h>

int main()
{
    const int m = 23, n = 23, k = 23;    /* some problem size */
    double a[m*k], b[k*n], c[m*n];      /* initialize later */
    libxsmm_dmm_function xmm = NULL;     /* function pointer */

    libxsmm_mm(m, n, k, a, b, c);        /* auto-dispatched */
    libxsmm_imm(m, n, k, a, b, c);       /* inlined */
    libxsmm_blasmm(m, n, k, a, b, c);    /* BLAS */
    libxsmm_dmm_23_23_23(a, b, c);      /* specialized */

    xmm = libxsmm_dmm_dispatch(23, 23, 23);
    if (xmm) {                            /* specialized */
        for (int i = 0; i < some; ++i) {
            xmm(a, b, c);                /* amortized */
        }
    }
}
```

# LIBXSMM: Getting Started (cont.)

## Usual mechanics

```
$ make ; make clean
```

```
$ make realclean
```

## Row major (default), or column-major

```
$ make ROW_MAJOR=0
```

## Specialization

```
$ make M="2 4" N="1" K="$(echo $(seq 2 5))"
```

Generates the following index set:

```
(2,1,2), (2,1,3), (2,1,4), (2,1,5),  
(4,1,2), (4,1,3), (4,1,4), (4,1,5)
```

# *LIBXSMM: Flexible Specialization*

Specialization using MNK variable (instead of M, N, and K)

```
$ make MNK="2 3, 23"
```

Generates the following index set:

```
(2, 2, 2), (2, 2, 3), (2, 3, 2), (2, 3, 3),  
(3, 2, 2), (3, 2, 3), (3, 3, 2), (3, 3, 3), (23, 23, 23)
```

## Background

- Takes a list of (grouped) indices (comma separated)
- Combines each group into all possible triplets

# *LIBXSMM: Automatic Code Dispatch*

## **Automatic code dispatch (levels)**

- 1. Below threshold ( $M \times N \times K \leq \text{LIBXSMM\_MAX\_MNK}$ )**
  - a) Specialized routine call (if available)**
  - b) Inlined code, or MKL DIRECT CALL**
- 2. Fallback code path (otherwise)**
  - c) LAPACK/BLAS call**

## **Adjusting the threshold (LIBXSMM\_MAX\_MNK)**

**\$ make THRESHOLD=\$((60 \* 60 \* 60))**

## **Adjusting the dispatch mechanism**

**\$ make SPARSITY=2**

# *LIBXSMM: Code Paths (Dispatch)*

- Supports any LAPACK/BLAS, optionally MKL DIRECT CALL
- Dispatch levels are avail. separately (customized dispatch)

`libxsmm_?imm`

`libxsmm_?blasmm`

- Amortizing dispatch cost (multiple calls of same M, N, K)

`libxsmm_?mm_dispatch`

- Specific kernel access e.g.,

`libxsmm_dmm_4_4_4`

# *LIBXSMM: Compile-time Tuning*

## Generate specific code path: AVX=1|2|3 or SSE=1

- Allows to speedup code generation when building the library; by default all code paths supported by the code generator are generated
- Allows to cross-build the library for a specific ISA extension; by default the compilation flag selects the actual code path according to -march=native

## Generate aligned store instructions: ALIGNED\_STORES=1

- Call side code must be prepared for round LDC up to be aligned

## Use of static information: LIBXSMM\_\* macros

- Allows for e.g., loop hints (LIBXSMM\_PRAGMA\_LOOP\_COUNT)

# *LIBXSMM: Compile-time Tuning (cont.)*

**ADVANCED:** directly invoking the assembly code generator

```
$ make generator
```

```
$ bin/generator
```

Generate Intrinsic code path: GENASM=0

- Usually not beneficial compared to default assembly code path
- Available for Intel Xeon Phi coprocessor and Intel AVX-512



# *LIBXSMM: Implementation*

## Highly optimized assembly code generation\*

- SSE3, AVX, AVX2, IMCI (KNCni), and AVX-512
- AVX-512 code quality
  - Maximizes number of immediate operands
  - Limits Instructions width to 16 Byte/cycle

## High level code optimizations

- Implicitly aligned leading dimension (LDC) – allows aligned store instr.
- Aligned load instructions (not yet exposed in the interface)
- Sophisticated data prefetch (not yet exposed in the interf.)

\* There is also a non-default Intrinsics code generation targeting IMCI/KNCni and AVX-512

# *LIBXSMM: Code Samples*

## **samples/smm: blas, dispatched, inlined, and specialized**

- Multiply a series of A and B matrices into a series of C matrices  
STREAMing A and B from memory, accumulates C likely in LLC\*
- Multiply two matrices into a destination matrix  
Likely operates entirely within the LLC\*

## **samples/cp2k: more complex code sample**

- STREAMing A and B (memory), accumulates C (likely in LLC\*)
- Flushes C from time to time (memory)

\* If the problem size fits into the Last-Level Cache (LLC)

# LIBXSMM: CP2K Kernels

## Background

- Based on LIBXSMM's "cp2k" code sample
  - Approximates CP2K core functions
  - Not (yet) exactly modeling CP2K e.g., no index array used
- CK2K performs a mixture of "computation" and "streaming"
  - Memory streaming of matrix operands using unaligned loads
  - Result accumulation using aligned store instructions (thread local)
  - Write-back of thread-local results into (co)processor's global memory
  - Minor synchronization overhead (either "FP atomics" or set of locks)

```
#!/bin/bash
```

```
make ROW MAJOR=0 \
ALIGNED_STORES=1 \
MNK=\
"23" \
"6" \
"14 16 29" \
"14 32 29" \
"5 32 13 24 26" \
"9 32 22" \
"32" \
"64" \
"78" \
"16 29 55" \
"32 29 55" \
"12" \
"13 26 28 32 45"
```

\* The Shell script builds LIBXSMM using a list of grouped indices which are combined into all possible triplets.

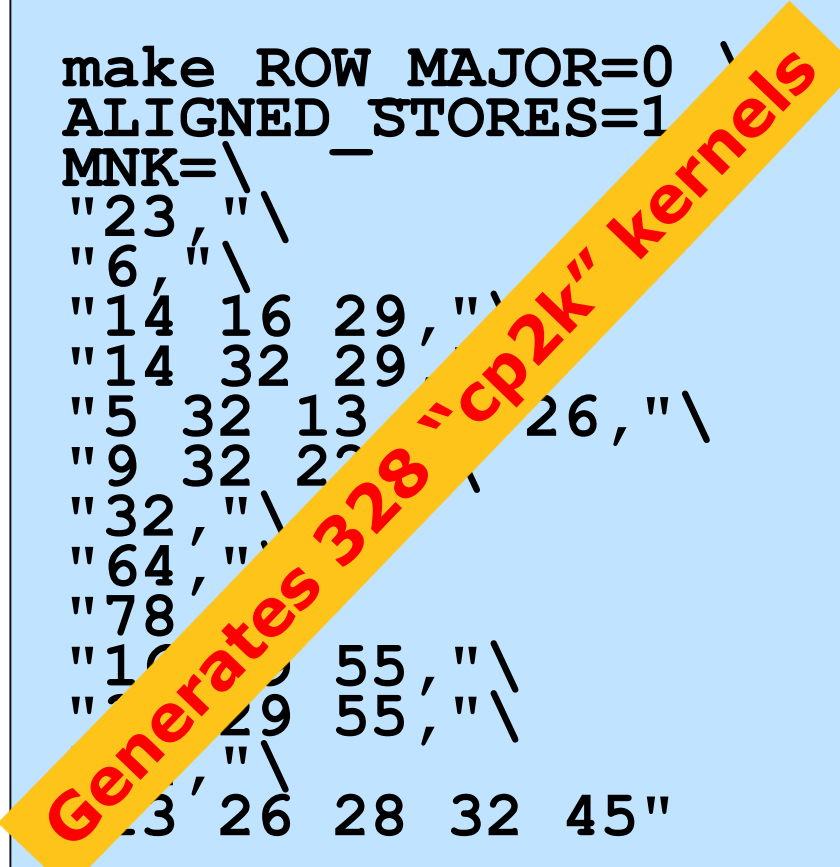
# LIBXSMM: CP2K Kernels

## Background

- Based on LIBXSMM's "cp2k" code sample
  - Approximates CP2K core functions
  - Not (yet) exactly modeling CP2K e.g., no index array used
- CK2K performs a mixture of "computation" and "streaming"
  - Memory streaming of matrix operands using unaligned loads
  - Result accumulation using aligned store instructions (thread local)
  - Write-back of thread-local results into (co)processor's global memory
  - Minor synchronization overhead (either "FP atomics" or set of locks)

```
#!/bin/bash

make ROW MAJOR=0 \
ALIGNED_STORES=1 \
MNK=\
"23" \
"6" \
"14 16 29" \
"14 32 29" \
"5 32 13 26" \
"9 32 27" \
"32" \
"64" \
"78" \
"10 29 55" \
"13 29 55" \
"13 26 28 32 45"
```



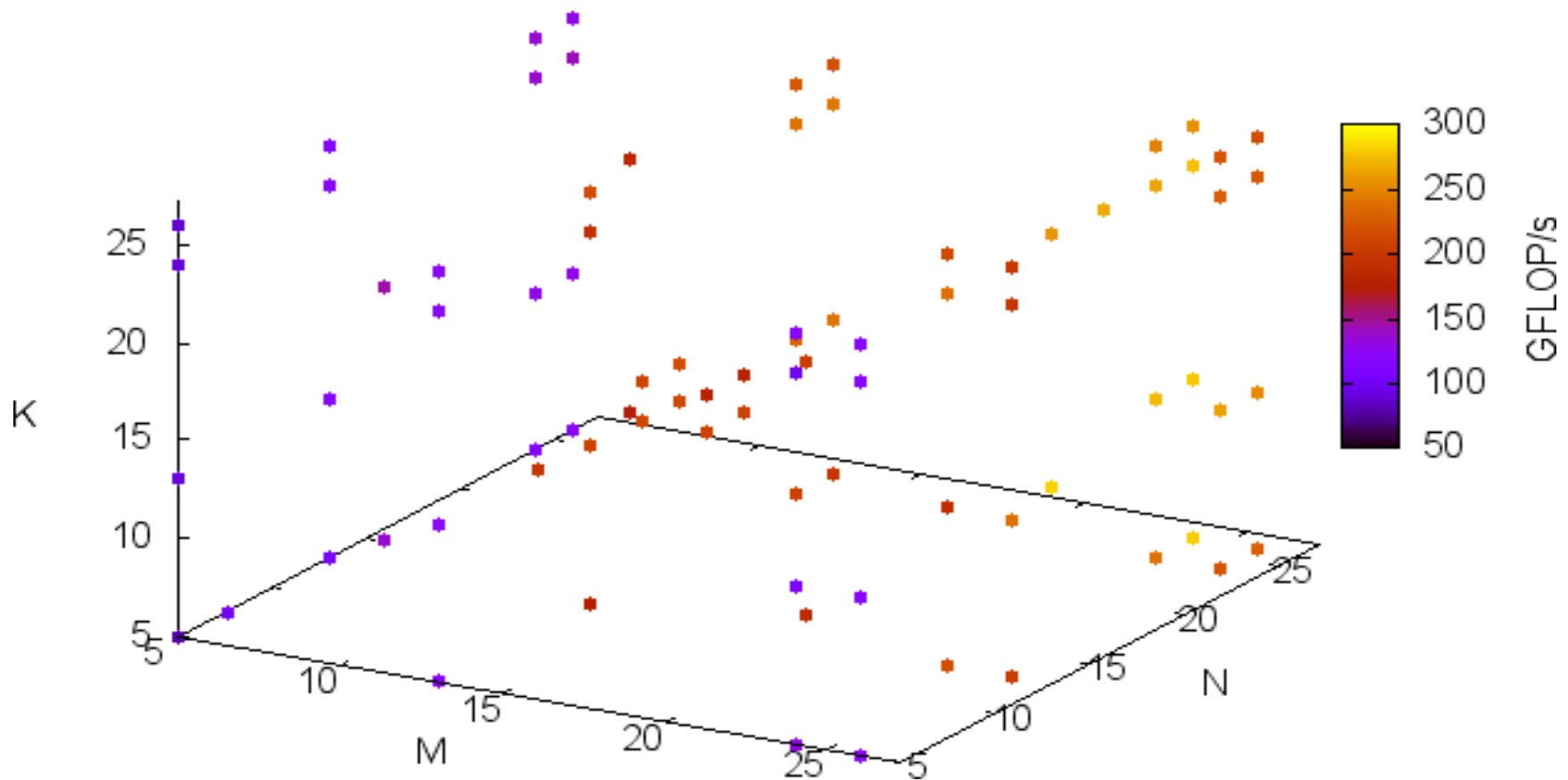
\* The Shell script builds LIBXSMM using a list of grouped indices which are combined into all possible triplets.

# *LIBXSMM Performance Results*

**Intel Xeon Phi 7120 Coprocessor ("KNC")**

# LIBXSMM: Performance of CP2K Kernels

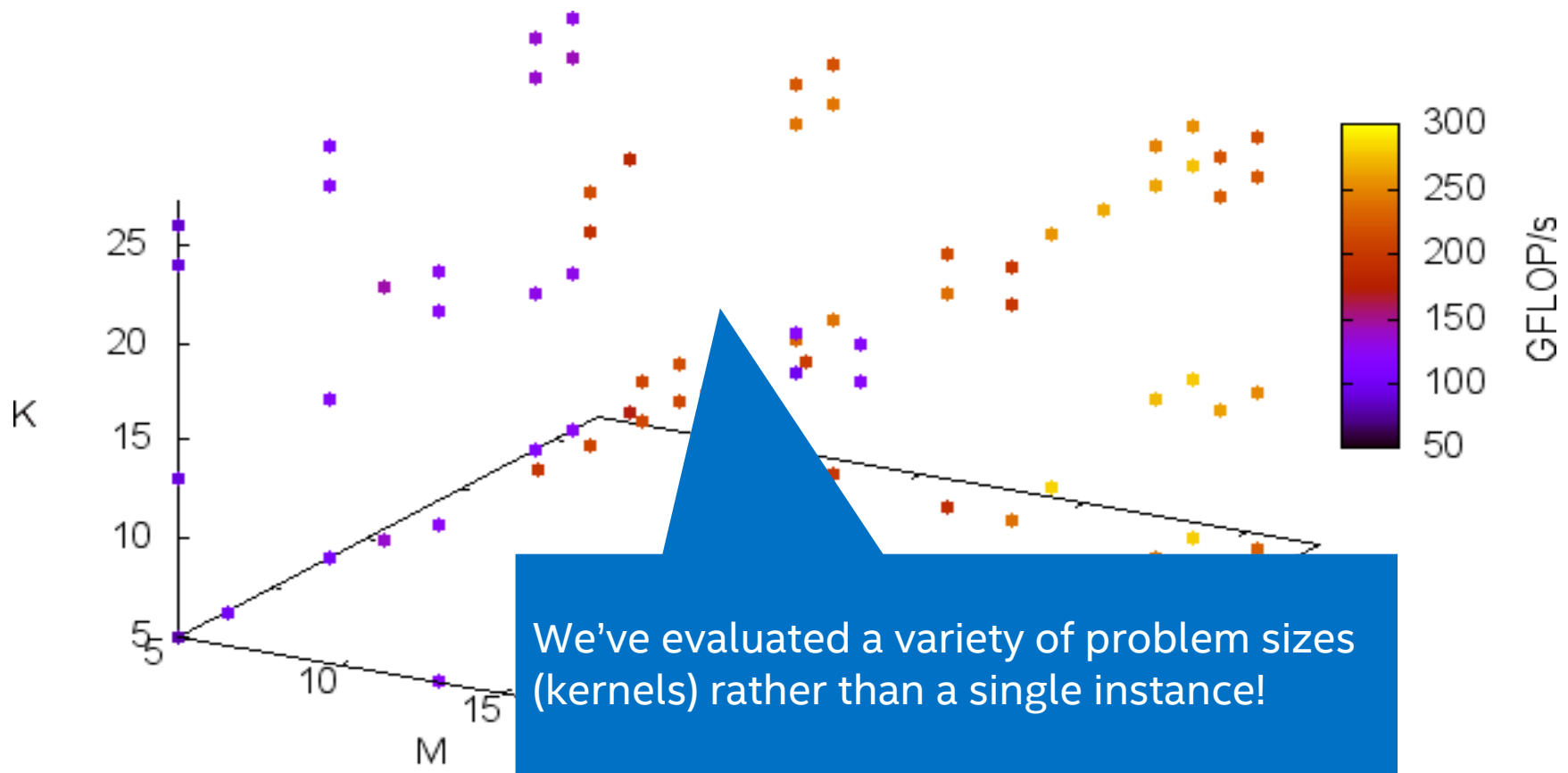
Parameter Space Exploration (subset of 328 kernels)



\* Intel Xeon Phi 7120 Coprocessor @ 1.2 GHz, 16 GB GDDR5, 240 threads with 4t/C using IMCI/KNCni

# LIBXSMM: Performance of CP2K Kernels

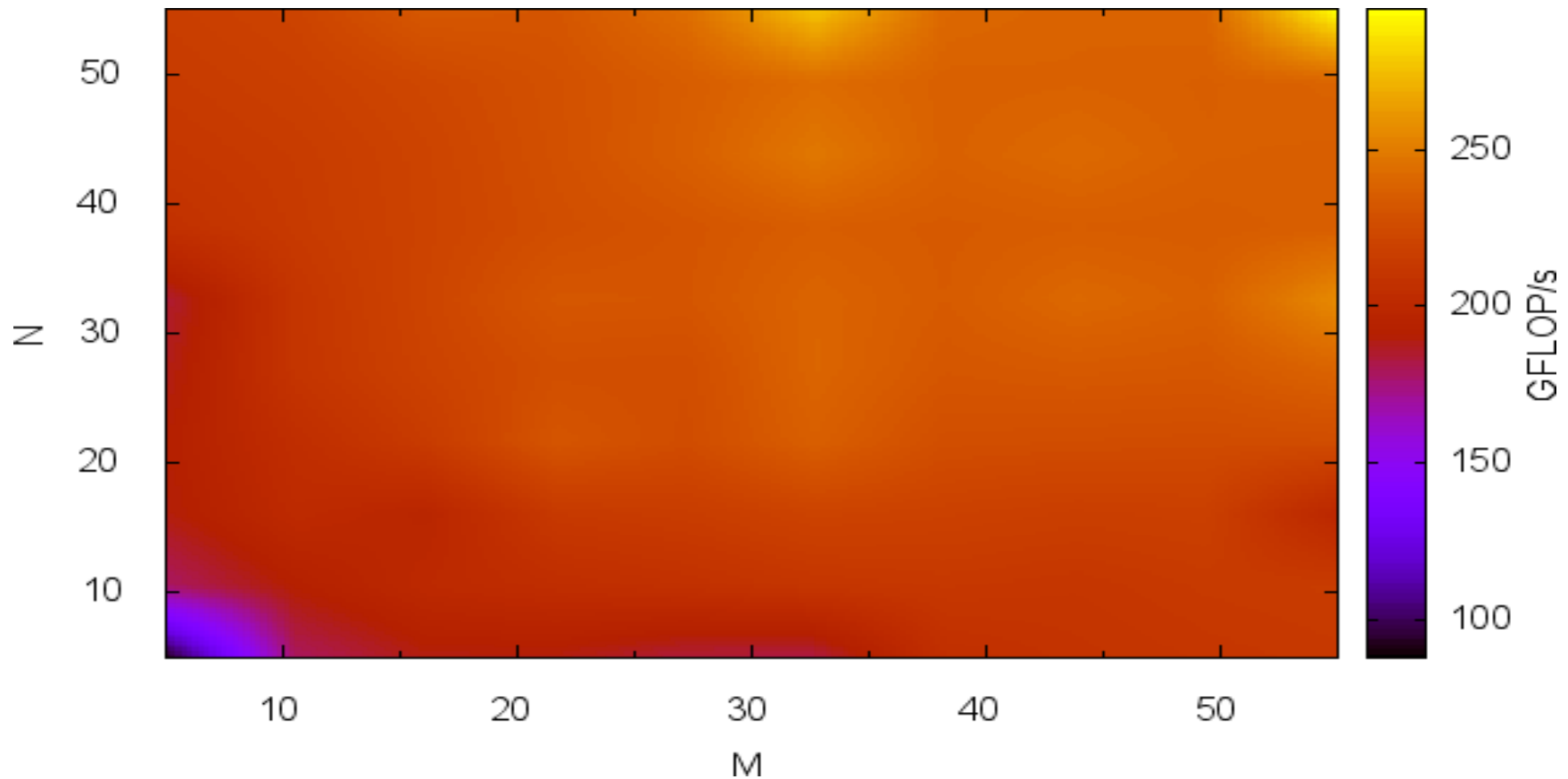
Parameter Space Exploration (subset of 328 kernels)



\* Intel Xeon Phi 7120 Coprocessor @ 1.2 GHz, 16 GB GDDR5, 240 threads with 4t/C running IMCI/KNCni code

# *LIBXSMM: Performance of CP2K Kernels*

*K-Average over MN-Parameter Space (328 kernels)*

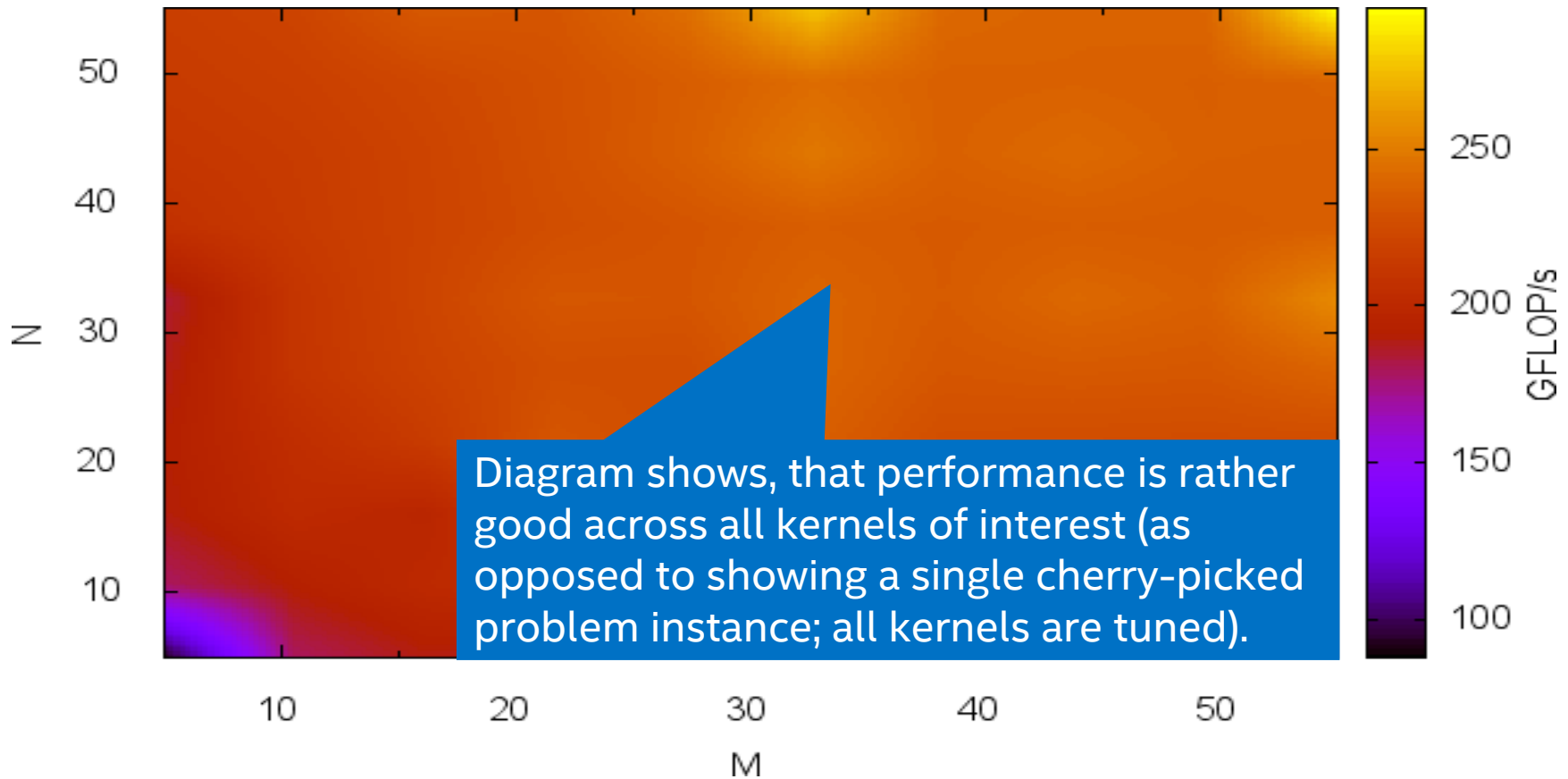


\* Intel Xeon Phi 7120 Coprocessor @ 1.2 GHz, 16 GB GDDR5, 240 threads with 4t/C running IMCI/KNCni code



# *LIBXSMM: Performance of CP2K Kernels*

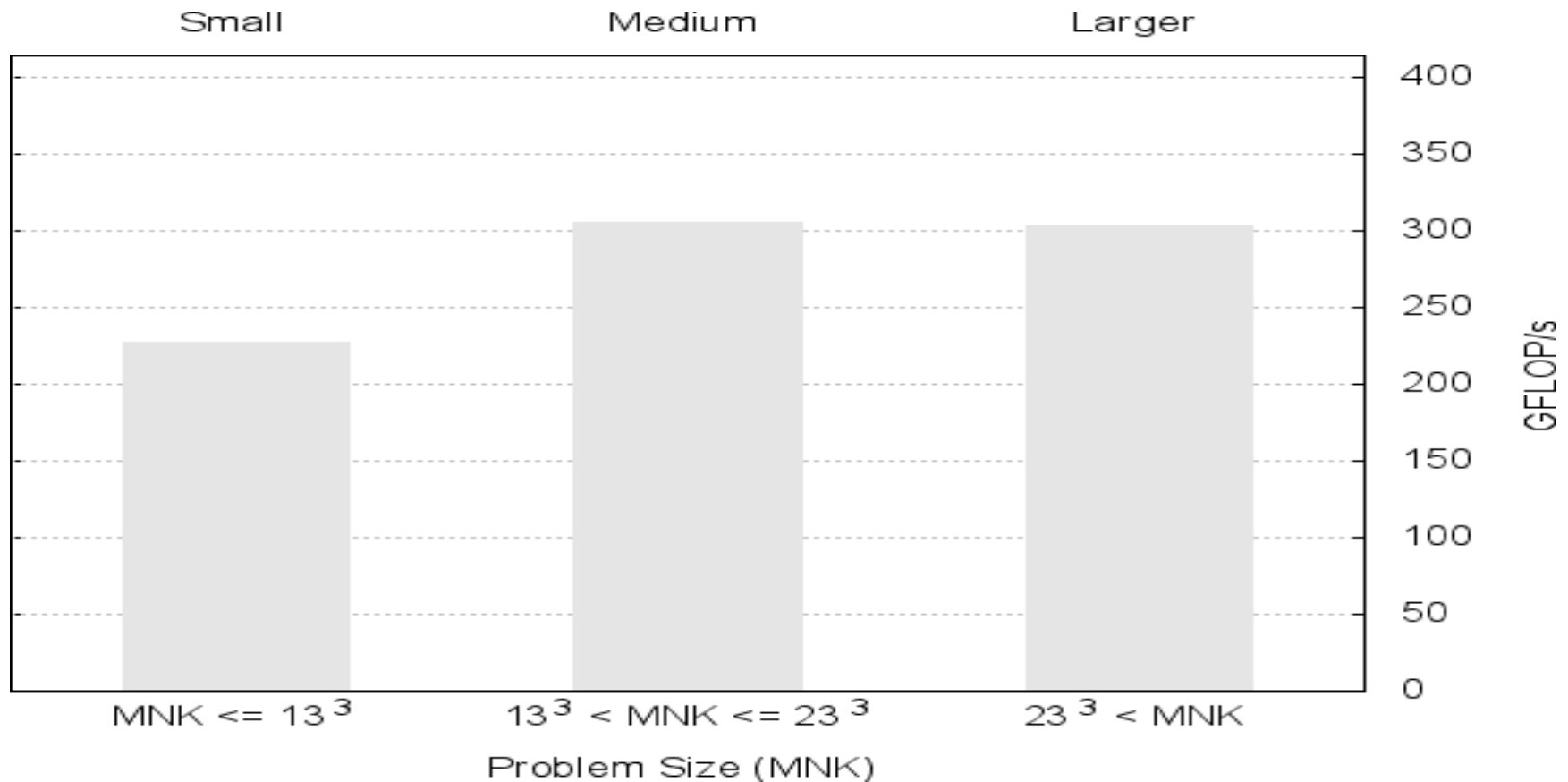
*K-Average over MN-Parameter Space (328 kernels)*



\* Intel Xeon Phi 7120 Coprocessor @ 1.2 GHz, 16 GB GDDR5, 240 threads with 4t/C running IMCI/KNCni code

# *LIBXSMM: Performance of CP2K Kernels*

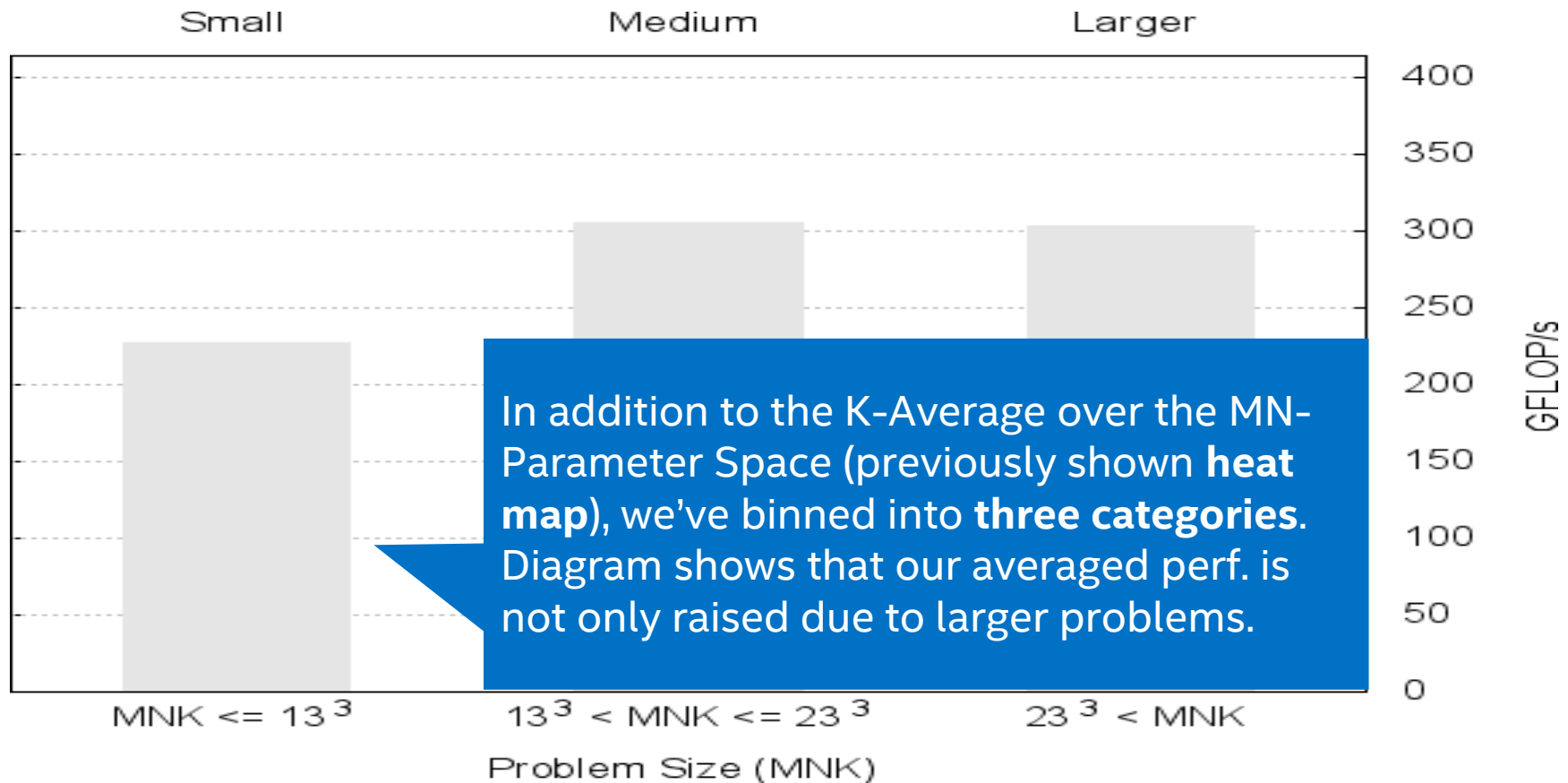
*Average per Bin of Problem Size (328 kernels)*



\* Intel Xeon Phi 7120 Coprocessor @ 1.2 GHz, 16 GB GDDR5, 240 threads with 4t/C running IMCI/KNCni code

# LIBXSMM: Performance of CP2K Kernels

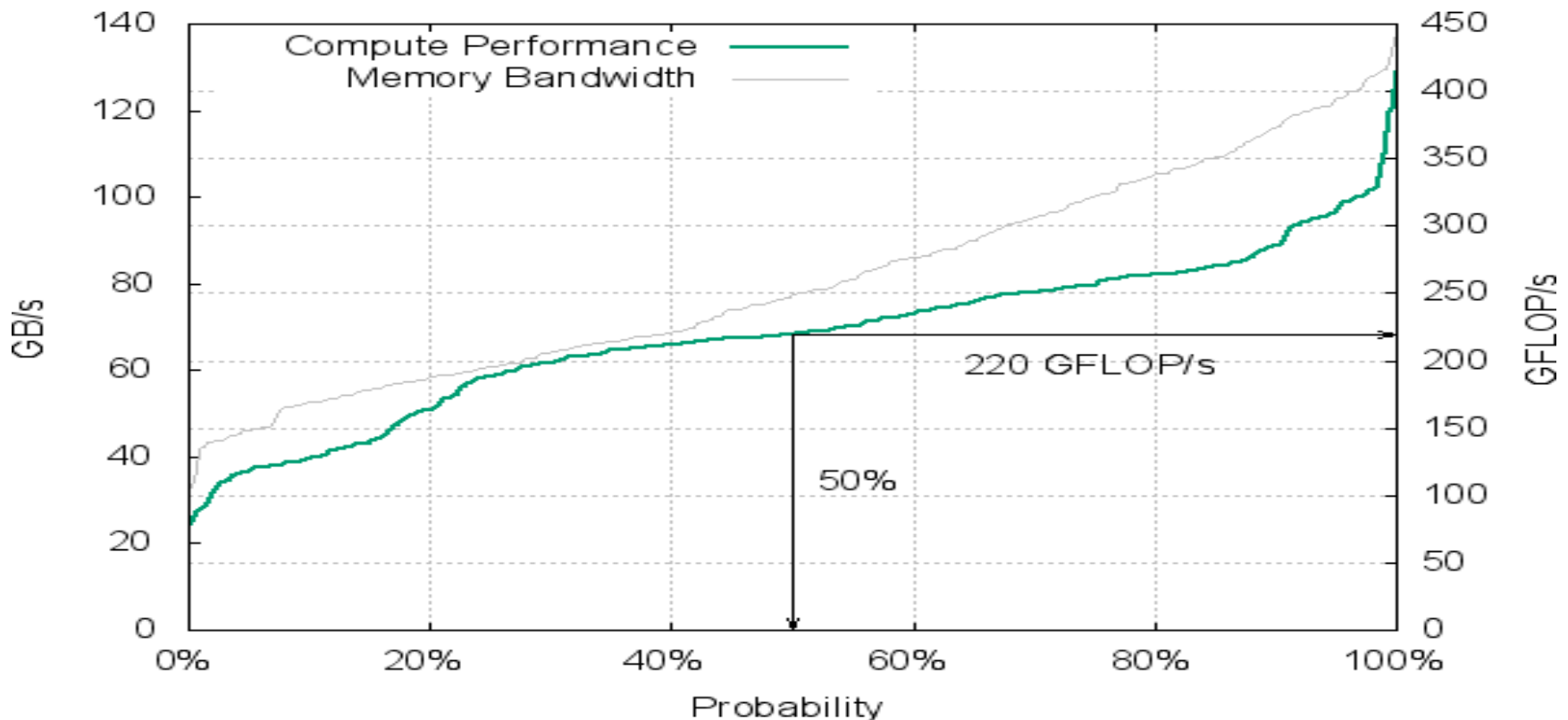
Average per Bin of Problem Size (328 kernels)



\* Intel Xeon Phi 7120 Coprocessor @ 1.2 GHz, 16 GB GDDR5, 240 threads with 4t/C running IMCI/KNCni code

# LIBXSMM: Performance of CP2K Kernels

## CDF – Cumulative Distribution Function (328 kernels)

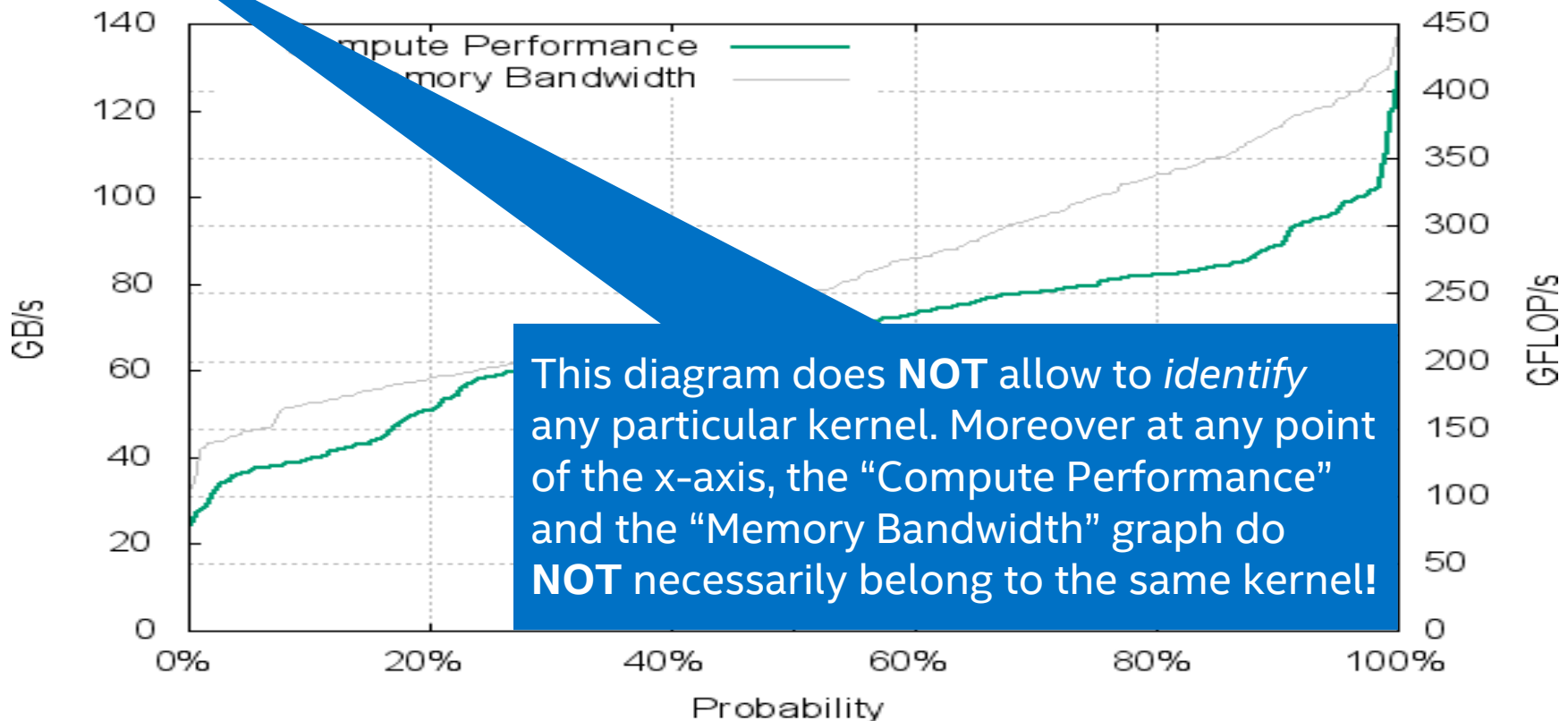


Minimum: 79 GFLOP/s Geo. Mean: 210 GFLOP/s Median: 220 GFLOP/s Maximum: 414 GFLOP/s

\* Intel Xeon Phi 7120 Coprocessor @ 1.2 GHz, 16 GB GDDR5, 240 threads with 4t/C running IMCI/KNCni code

# LIBXSMM: Performance of CP2K Kernels

CDF – Cumulative Distribution Function (328 kernels)



Minimum: 79 GFLOP/s Geo. Mean: 210 GFLOP/s Median: 220 GFLOP/s Maximum: 414 GFLOP/s

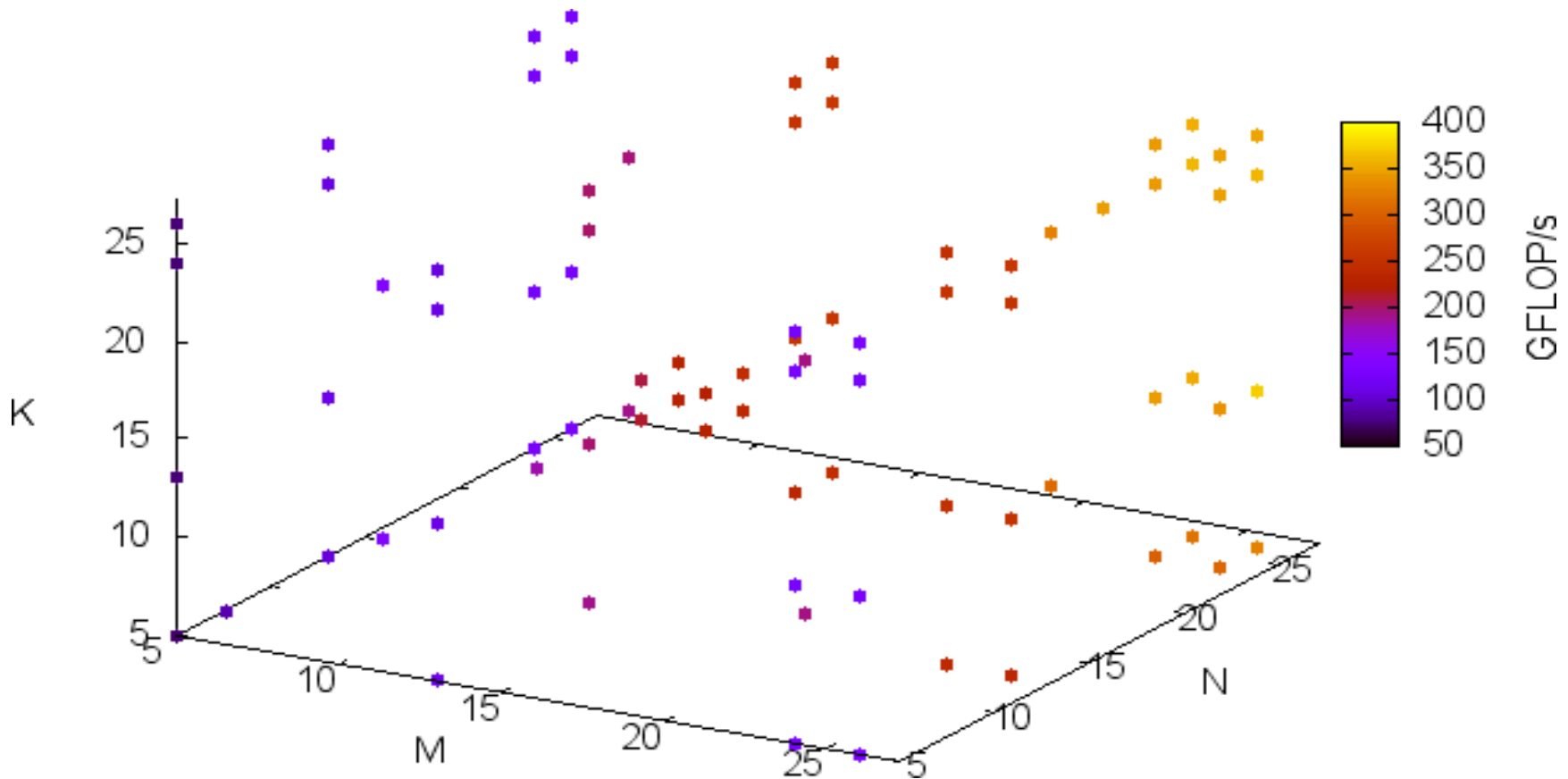
\* Intel Xeon Phi 7120 Coprocessor @ 1.2 GHz, 16 GB GDDR5, 240 threads with 4t/C running IMCI/KNCni code

# *LIBXSMM Performance Results*

**Intel Xeon E5-2699v3 ("Haswell")**

# LIBXSMM: Performance of CP2K Kernels

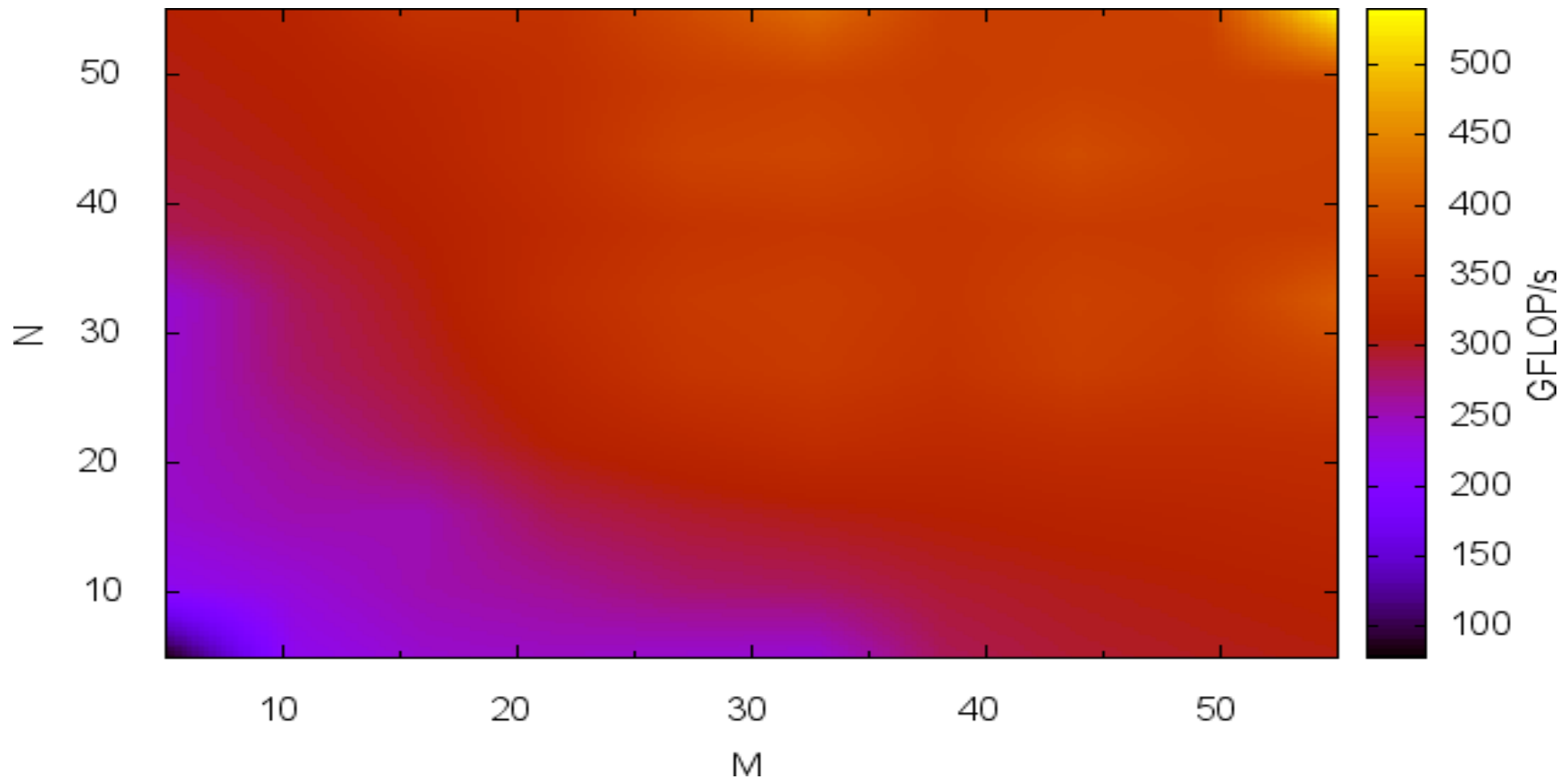
Parameter Space Exploration (subset of 328 kernels)



\* Intel Xeon E5-2699v3 @ 2.3 GHz, 72 threads with 2t/C running AVX2 code

# *LIBXSMM: Performance of CP2K Kernels*

*K-Average over MN-Parameter Space (328 kernels)*

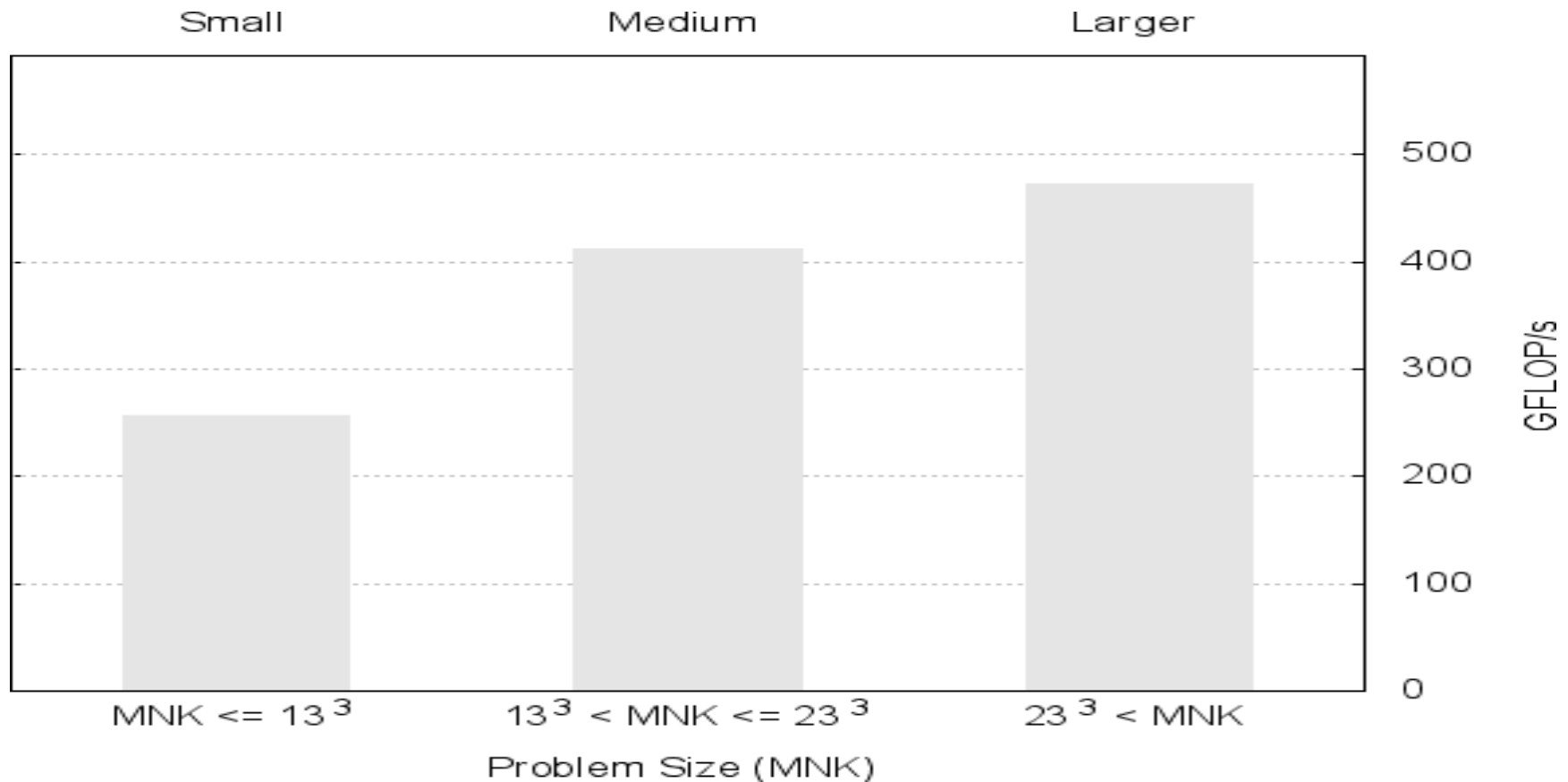


\* Intel Xeon E5-2699v3 @ 2.3 GHz, 72 threads with 2t/C running AVX2 code



# LIBXSMM: Performance of CP2K Kernels

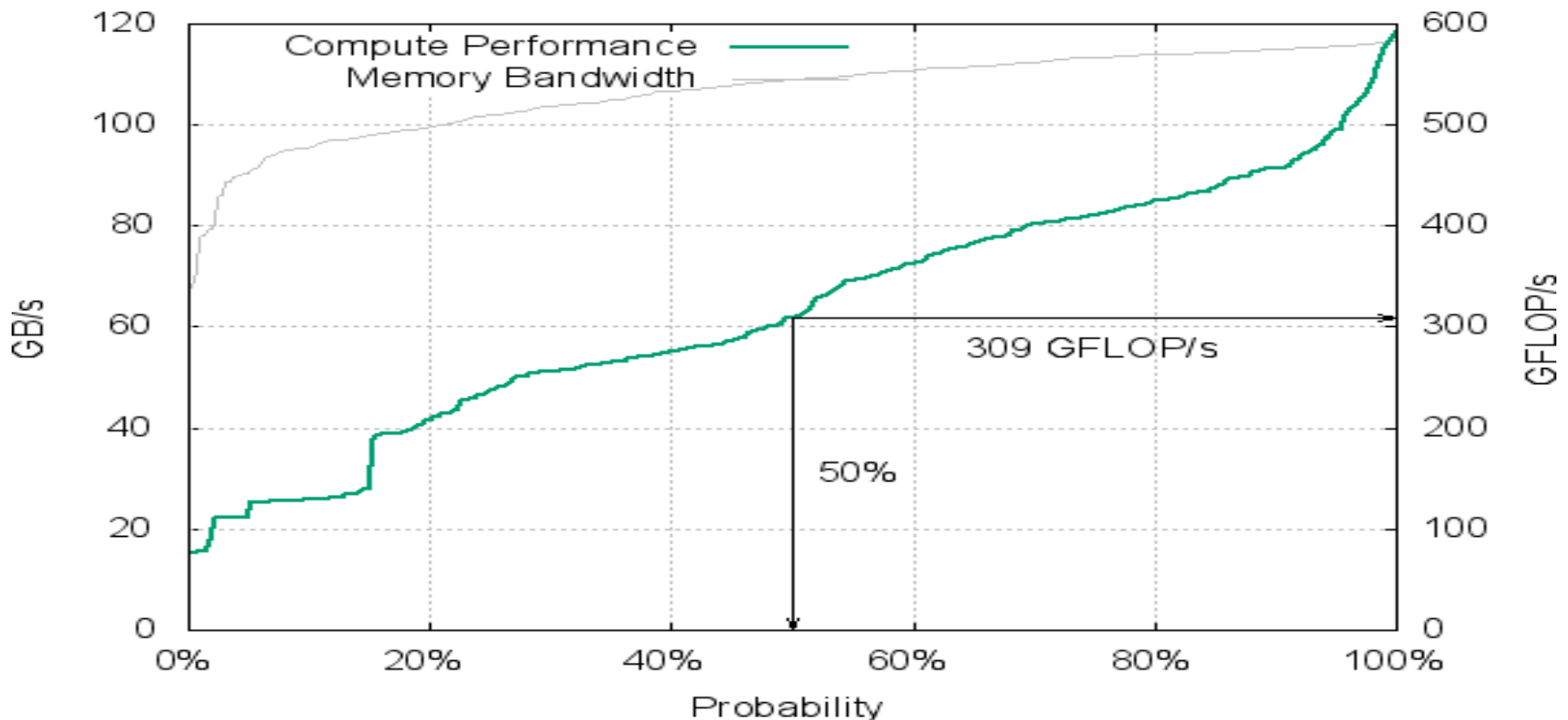
Average per Bin of Problem Size (328 kernels)



\* Intel Xeon E5-2699v3 @ 2.3 GHz, 72 threads with 2t/C running AVX2 code

# LIBXSMM: Performance of CP2K Kernels

## CDF – Cumulative Distribution Function (328 kernels)



Minimum: 76 GFLOP/s Geo. Mean: 288 GFLOP/s Median: 309 GFLOP/s Maximum: 592 GFLOP/s

\* Intel Xeon E5-2699v3 @ 2.3 GHz, 72 threads with 2t/C running AVX2 code

# *LIBXSMM: Roadmap*

- Full xGEMM interface\*, and native FORTRAN interface
- Just-in-Time (JIT) runtime dynamic code generation
- API supporting sparse matrices and other cases

\* Optionally intercepting xGEMM calls (LD\_PRELOAD).

# *LIBXSMM: Applications*

**[1] <http://cp2k.org/>:** Open Source Molecular Dynamics application which is able to use LIBXSMM; see <https://github.com/cp2k/cp2k/tree/intel>.

**[2] <http://www.seissol.org/>:** SeisSol is one of the leading codes for earthquake scenarios, in particular for simulating dynamic rupture processes. LIBXSMM provides highly optimized assembly kernels which form the computational back-bone of SeisSol; see [https://github.com/TUM-I5/seissol\\_kernels/tree/lts\\_compressed](https://github.com/TUM-I5/seissol_kernels/tree/lts_compressed).

# References

## LIBXSMM home page

<https://github.com/hfp/libxsmm>

## Related material

### **[1] Code generator for matrix-matrix multiplications**

<https://github.com/TUM-I5/GemmCodeGenerator>

### **[2] Performance engineering and code tuning (video/slide series)**

[http://user.cscs.ch/support/tutorials/2014/node\\_level\\_performance\\_engineering\\_15\\_16\\_may\\_2014/index.html](http://user.cscs.ch/support/tutorials/2014/node_level_performance_engineering_15_16_may_2014/index.html)

### **[3] Optimized matrix transposes**

<http://research.colfaxinternational.com/post/2013/04/25/Transposition-Xeon-Phi.aspx>

## Intel collaterals

### **[4] Xeon Phi Applications and Solutions Catalog**

<http://software.intel.com/xeonphicatalog>

### **[5] 3rd Party Tools and Libraries**

<https://software.intel.com/en-us/articles/intel-and-third-party-tools-and-libraries-available-with-support-for-intelr-xeon-phi>

# 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.

Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

## 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

