# Kokkos Autotuning for SpMV Kernels on High-Performance Accelerators

**Simon Garcia De Gonzalo[1], Simon Hammond[2], Christian Trott[2], Wen-Mei Hwu[1]**
**[1]University of Illinois at Urbana-Champaign, [2]Sandia National Laboratories**

University of Illinois at Urbana-Champaign

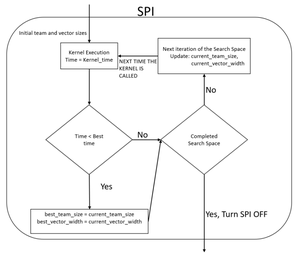
Sandia National Laboratories

## Motivation & Contributions

- Kokkos provide tuning parameters such as the size of thread teams and vector width
- For many applications, choosing the right parameters can be highly dependent on the data input or application configuration.
- SpMV performance is often highly correlated with the associated matrix sparsity
- Extend the current set of Kokkos profiling tools with an online-autotuner that can iterate over possible:
  - Thread-team size
  - Vector width
- Evaluate the autotuner on the latest classes of HPC devices –NVIDIA's Pascal GP100, and Intel's Knights Landing (KNL)

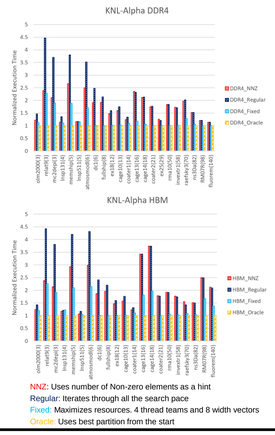
## Autotuner

- Implemented using the KokkosP performance hooks interface.
  - Dynamically loaded
  - Collects rich information about parallel regions
  - Support for V-tune and Nsight
- Uses runtime information provided by the hooks as feedback to improve the parameters




- User must first register the parameters using the registration API
  - line 8 of code listing
- SPI responsible for iterating through possible combinations of parameters
  - Each hardware platform has a different search space
  - Hinges on iterative applications
  - Number of non-zeros can be given to trim the search space

## Knights Landing Results




NNZ: Uses number of Non-zero elements as a hint
Regular: Iterates through all the search pace
Fixed: Maximizes resources. 4 thread teams and 8 width vectors
Oracle: Uses best partition from the start

Autotuner:

- DDR4:
  - 2.1X to 4.4X slower than the Oracle for large sparse matrices but very few non-zero elements. NNZ hint improves the above difference to 1.9X to 2.6X.
    - Most optimal setup for these type of Matrices is 1 thread per team and vector width of 1
    - Larger team sizes and vector sizes suffer from poor SIMD compute to L1 and L2 access.
  - 1.5X to 2.0X slower that the Oracle for smaller and slightly denser matrices
    - Suboptimal choices do not incur large penalties
    - Fixed is one of the most optimal choices for this type of matrices
  - 13% to 45% performance loss compared to the Oracle for very large and dense matrices
    - Bandwidth bound compare to compute bound.
    - Maximize compute resources becomes less necessary to the overall performance.
- HBM:
  - Similar to DDR4 except for large denser matrices
    - Bandwidth drastically increases from 90GB/s to 400GB/s
    - Bandwidth no longer primary performance bottleneck

## Kokkos

- Programing model that abstract code from the finer intricacies of hardware details
- Provides abstractions for the memory space and execution space
- Parallel patterns:
  - parallel-for, -reduce, and -scan
- Abstract machine model
  - Multiple execution and memory spaces

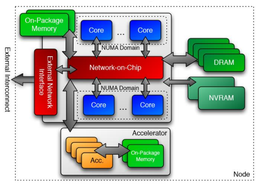



Fig. 1. Conceptual Model of a High Performance Computing Node [9]

## Accelerators and Matrices

ATTRIBUTES OF EACH OF THE MATRICES THAT WERE USED. THE NUMBER IN PARENTHESIS AFTER THE NAME IS THE AVERAGE NUMBER OF NON-ZERO ELEMENTS PER ROW. THE REST OF THE TABLE SHOWS THE NUMBER OF ROWS, COLUMNS AND THE TOTAL NUMBER OF NON-ZERO ELEMENTS IN THE MATRIX.

| Matrix Name | Number of Rows | Number of Clums | Non-Zeros |
|---|---|---|---|
| olm2000(3) | 2,000 | 2,000 | 7,996 |
| relat9(3) | 12,360,060 | 549,336 | 38,955,420 |
| mc2depi(3) | 525,825 | 525,825 | 2,100,225 |
| lnsp131(4) | 131 | 131 | 536 |
| memchip(5) | 2,707,524 | 2,707,524 | 14,810,202 |
| lnsp511(5) | 511 | 511 | 2,796 |
| atmosmodl(6) | 1,489,752 | 1,489,752 | 10,319,760 |
| dc1(6) | 116,835 | 116,835 | 766,396 |
| fullship(8) | 2,987,012 | 2,987,012 | 26,621,990 |
| ex18(12) | 5,773 | 5,773 | 71,805 |
| cage10(13) | 11,397 | 11,397 | 150,645 |
| coater1(14) | 1,348 | 1,348 | 19,457 |
| cage13(16) | 445,315 | 445,315 | 7,479,343 |
| cage14(18) | 1,505,785 | 1,505,785 | 27,130,349 |
| coater2(21) | 9,540 | 9,540 | 207,308 |
| ex25(29) | 848 | 848 | 24,612 |
| rma10(50) | 46,835 | 46,835 | 2,374,001 |
| invextr1(58) | 30,412 | 30,412 | 1,793,881 |
| raefsky3(70) | 21,200 | 21,200 | 1,488,768 |
| ns3Da(82) | 20,414 | 20,414 | 1,679,599 |
| RM07R(98) | 381,689 | 381,689 | 37,464,962 |
| fluorem(140) | 2,017,169 | 2,017,169 | 283,073,458 |

SEARCH-SPACE BOUNDS PER ARCHITECTURE: THE LIMITS FOR EACH DEVICE WERE DETERMINED BY RUNNING A LARGE NUMBER OF SPMV PROBLEMS. THE RESULTING BOUNDS NICELY MATCH HARDWARE CHARACTERISTICS OF EACH DEVICE. IT IS IMPORTANT TO NOTE THAT THE SEARCH SPACE FOR THE KNL IS ABOUT 3 TIMES LARGER THAN THE PASCAL SEARCH SPACE

| Architecture | Max Team Size | Min Team Size | Max Vector Width | Min Vector Width |
|---|---|---|---|---|
| Pascal | 1024 | 32 | 32 | 1 |
| Knights Landing | 2048 | 1 | 16 | 1 |

Fig. 4. Distribution of non-zero elements for atmosmodl. Size: 1,489,752x1,489,752. Number of non-zeros: 10,319,760. [18]

Fig. 5. Distribution of non-zero elements for cage10. Size: 11,397x11,397. Number of non-zeros: 150,645.[14]

Fig. 6. Distribution of non-zero elements for lnsp131. Size: 131x131. Number of non-zeros: 536. [14]

## Autotuner selection

BEST-PERFORMING SELECTION BY THE AUTOTUNER

| Matrix | # Of Non Zeros Per-row | Pascal | | KNL Alpha DDR4 | | KNL Alpha HBM | | KNL Delta | |
|---|---|---|---|---|---|---|---|---|---|
| | | Team Size | Vector Width | TeamSize | Vector Width | Team Size | Vector Width | Team Size | Vector Width |
| relat9 | 3 | 128 | 2 | 1 | 1 | 1 | 2 | 1 | 4 |
| lnsp131 | 4 | 32 | 32 | 16 | 2 | 16 | 1 | 16 | 4 |
| atmosmodl | 6 | 64 | 4 | 1 | 4 | 1 | 1 | 1 | 1 |
| fullship | 8 | 32 | 32 | 4 | 1 | 4 | 4 | 4 | 1 |
| cage10 | 13 | 32 | 4 | 4 | 16 | 2 | 8 | 4 | 1 |
| coater1 | 14 | 32 | 8 | 16 | 2 | 2048 | 8 | 2048 | 1 |
| coater2 | 21 | 64 | 8 | 2048 | 16 | 4 | 2 | 4 | 4 |
| rma10 | 50 | 32 | 16 | 4 | 4 | 2 | 4 | 2 | 4 |
| raefsky | 70 | 32 | 16 | 2 | 8 | 4 | 1 | 2 | 8 |
| fluorem | 140 | 32 | 32 | 8 | 16 | 1 | 2 | 1 | 8 |

- Each architecture had a substantially different set of optimal parameters
- The two KNL configurations have the same parameters less than half the time
- For the most optimal application performance, the programmer would have to re-tune his or her application when porting it between configurations of the same architecture.
- GPU's shows a preference for a large thread team size when compared to the KNL type devices
- KNL Alpha using only theDDR4 memory, and KNL Delta the vector width creases as the as the number of non-zeros per row increases
- Not particularly true for KNL Alpha with HBM enable as it may depend more heavily on other matrix features such as matrix size.
- Portability across and within architectures has to be done by taking into account small hardware details and data characteristics

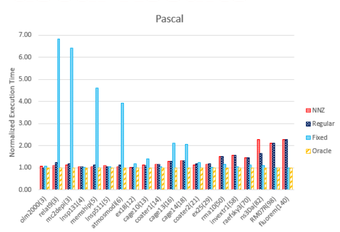
## SPVM Skeleton code

- The algorithm is defined within a C++ functor from line 19 to 32
- Series of nested parallel patterns in lines 19 and 22.
- Functor is instantiated in line 5 and used as the last parameter for the parallel pattern in line 11.
  - Breaking the problem size into groups of team threads and specifies the vector width

Listing 1. Kokkos SPMV skeleton code

```
int team_size;                                                          1
int vector_length;                                                      2
                                                                        3
// instantiate SPMV functor                                             4
SPMV_Functor<...> func (...);                                           5
                                                                        6
// registration of the team_size and vector_length parameters           7
Kokkos::Profiling::autoTune(&team_size,&vector_length);                 8
                                                                        9
//These parameters are use by the Kokkos TeamPolicy to map to hardware  10
Kokkos::parallel_for("SPMV",                                            11
                     Kokkos::TeamPolicy<Kokkos::Schedule<Kokkos::Dynamic>>  12
                                (league_size,team_size,vector_length),func);  13
// ...                                                                  14
struct SPMV_Functor {                                                   15
// ...                                                                  16
  operator() (const team_member& dev) const                             17
  {                                                                     18
    Kokkos::parallel_for(Kokkos::TeamThreadRange(dev,0,rows_per_team),[=](...){  19
      // ...                                                           20
      //perform vector reduction                                      21
      Kokkos::parallel_reduce(Kokkos::ThreadVectorRange(dev,row_length),[=](...){  22
        // ...                                                         23
        lsum += ...;                                                  24
      },sum);                                                         25
                                                                      26
      // Add the results once per thread                              27
      Kokkos::single(Kokkos::PerThread(dev), [&] () {                 28
          //                                                          29
          my_y(iRow) = sum ;                                          30
      });                                                             31
    });                                                               32
  }                                                                   33
// ...                                                                34
}                                                                     35
```

## Pascal Results




NNZ: Uses number of Non-zero elements as a hint
Regular: Iterates through all the search pace
Fixed: Maximizes resources. 32 by 32 CUDA blocks
Oracle: Uses best partition from the start

- Fixed:
  - Large irregular matrices perform poorly due to divergence
  - Smaller irregular matrices and denser matrices perform almost optimally
  - By around 50 non-zero elements divergence is not longer an issue
- Autotunner
  - 3.5x to 5X faster than fixed for large irregular matrices and only 12% to 13% slower than Oracle
  - As the number of non-zero increase performance is only marginally better than Fixed
  - For the denser type of matrices sub-optimal choices by the autotunner suffer substantial penalties
  - NNZ hint shows mixed results

## Conclusion & Future work

- Extended the Kokkos performance tools with an autotuner that iterates over possible candidate parameters.
- Compared the autotuner against a Fixed approach and the Oracle on the latest two distinct accelerator architectures available to this date.
- Identified matrix characteristics that affect the performance of the autotuners.
- Plan to augment the current autotuner with capabilities to extract information about the matrix and prune the search space using more advance heuristics

## Acknowledgments

Sandia National Laboratories