Strassen algorithm. Advanced optimization for Intel(R) Xeon Phi(TM)

Oleg Shapovalov

Efim Sergeev

Victor Getmansky

Dmitry Kryzhanovsky

Singularis Lab, Ltd.

Volgograd State Technical University

Strassen algorithm

The usual number of scalar operations (i.e., the total number of additions and multiplications) required to perform matrix multiplication is

$$M(n) = 2n^3 - n^2$$

However, Strassen (1969) discovered how to multiply two matrices in

$$S(n) = 7n^{\log_2 7} - n^2 \approx 7n^{2.81} - n^2$$

References

- Bailey, D., Lee, K., Simon, H.: Using Strassen's Algorithm to Accelerate the Solution of Linear Systems. J. of Supercomputing 4(4), 357-371 (1991)
- Dumitrescu, B.: Improving and estimating the accuracy of Strassen's algorithm. Numer. Math. 79, 485-499 (1998)
- ElGindy, H., Ferizis, G.: On Improving the Memory Access Patterns During The Execution of Strassen's Matrix Multiplication Algorithm. ACSC '04 Proc. of the 27th Australasian conf. on Computer science – Vol. 26, 109-115 (2004)
- Li, J., Ranka, S., Sahni, S.: Strassen's Matrix Multiplication on GPUs. ICPADS '11 Proc. of the 2011 IEEE 17th International Conf. on Parallel and Distributed Systems, 157-164 (2011)
- Lipshitz, B., Ballard, G., Demmel, J., Schwartz, O.: Communication-avoiding parallel strassen: implementation and performance. SC '12 Proc. of the International Conf. on High Performance Computing, Networking, Storage and Analysis, Art. 101 (2012)
- Heinecke, A.: Evaluation of DGEMM Implementation on Intel Xeon Phi Coprocessor. J. of Computers, Vol 9, № 7 (2014), 1566-1571 (2014)
- Intel® Threading Challenge 2009. 2.1 Strassen Matrix Multiplication

Strassen algorithm

$$C = A * B = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} =$$

$$= \begin{bmatrix} S_1 + S_2 + S_3 - S_4 & S_4 + S_6 \\ S_3 + S_5 & S_1 - S_5 + S_6 + S_7 \end{bmatrix}$$

$$S_1 = (A_{00} - A_{11}) * (B_{00} + B_{11})$$
$$S_2 = (A_{01} - A_{11}) * (B_{10} + B_{11})$$
$$S_3 = A_{11} * (B_{10} + B_{00})$$
$$S_4 = (A_{00} + A_{01}) * B_{11}$$
$$S_5 = (A_{10} + A_{11}) * B_{00}$$
$$S_6 = A_{00} * (B_{01} - B_{11})$$
$$S_7 = (A_{10} - A_{00}) * (B_{00} - B_{01})$$

Connection to cluster

```
putty, winscp
ssh tornado.hpc.susu.ac.ru
user: userX
password: ******
```

```
http://supercomputer.susu.ru/users/support/
```

Serial version

- Folder 1
- build_x64.sh, build_mic.sh – building scripts
- matrix_strassen – wrappers for Strassen and Intel(R) MKL DGEMM
- matrix_strassen – Strassen implementation
- matrix_utils – matrix operations routines
- run – parsing command line, launch multiplication
- test_utils – testing execution time of matrix multiplication algorithms
- unit_tests – unit tests
- utils – memory routines

```
user1@login:~/1/src
$ ls
CUnit        matrix_mult.c      matrix_strassen.h  run.c         unit_tests.c
extra_libs   matrix_mult.h      matrix_utils.c     test_utils.c  utils.c
Makefile     matrix_strassen.c  matrix_utils.h     test_utils.h  utils.h
```

Verification

- Compare with Intel(R) MKL DGEMM

$$\varepsilon_{abs} = \max_{i,j \in 1..N} \left| a_{i,j} - b_{i,j} \right|$$

$$\varepsilon_{rel} = \max_{i,j \in 1..N} \frac{\left| a_{i,j} - b_{i,j} \right|}{\left| b_{i,j} \right|}$$

- Test succeeded if:

$$max(\varepsilon_{abs}, \varepsilon_{rel}) < 10^{-10}$$

Time measurement (test_utils.c)

- second/dsecnd - returns elapsed CPU time in seconds.

```
double run_method(MatrixMultiplicationMethod mm_method,
                  int n, double *A, double *B, double *C)
{
        int runs = 5;
        mm_method(n, A, B, C);

        double start_time = dsecnd();
        int i;
        for (i = 0; i < runs; ++i)
                mm_method(n, A, B, C);

        double end_time = dsecnd();
        double elapsed_time = (end_time - start_time) / runs;
        return elapsed_time;
}
```

matrix_strassen.c

```
void strassen_1( int n, double *A, int lda,
                        double *B, int ldb,
                        double *C, int ldc,
                                   int threshold)
{
      if (n <= threshold)
      {
            mkl_simple_ld(n, A, lda, B, ldb, C, ldc);
            return;
      }

…
```

matrix_strassen.c

```
int i, j;
for (i = 0; i < 2; ++i)
for (j = 0; j < 2; ++j)
{
      sub_a[i][j] = &A[i * n2 * lda + j * n2];
      sub_b[i][j] = &B[i * n2 * ldb + j * n2];
      sub_c[i][j] = &C[i * n2 * ldc + j * n2];
}
```

$$\begin{pmatrix} \boldsymbol{a} & a & \boldsymbol{a} & a \\ a & a & a & a \\ \boldsymbol{a} & a & \boldsymbol{a} & a \\ a & a & a & a \end{pmatrix}$$

matrix_strassen.c

$$M_6 = (A_{10} - A_{00}) * (B_{00} - B_{01})$$

```
…
matrix_sub_ld( n2, sub_a[1][0], lda,
                   sub_a[0][0], lda,
                                sub_c[0][1], ldc);

matrix_sum_ld( n2, sub_b[0][0], ldb,
                    sub_b[0][1], ldb,
                    sub_c[1][0], ldc);

strassen_1( n2, sub_c[0][1], ldc,
                sub_c[1][0], ldc,
                sub_c[1][1], ldc, threshold, n_threads); // m6

…
```

Intel(R) Math Kernel Library

$$C = \alpha \cdot op(A) + \beta \cdot op(B)$$

```
mkl_domatadd('R', 'N', 'N', n, n, 1.0,
                            A, lda, 1.0, B, ldb, C, ldc);
```

$$C = \alpha \cdot A \cdot B + \beta \cdot C$$

```
double alpha = 1.0;
double beta = 0.0;
cblas_dgemm( CblasRowMajor, CblasNoTrans, CblasNoTrans,
              n, n, n, alpha, a, lda, b, ldb, beta, c, ldc);
```

```
mkl_malloc, mkl_free, mkl_peak_mem_usage, ..
```

Build and run

```
./build_x64.sh
./build_mic.sh

chmod +x build_mic.sh – разрешить выполнение

salloc – выделить узел
squeue – просмотр выделенных узлов

ssh ps-micX – зайти на mic

exit – выйти с mic

scancel job_id – снять задачу
```

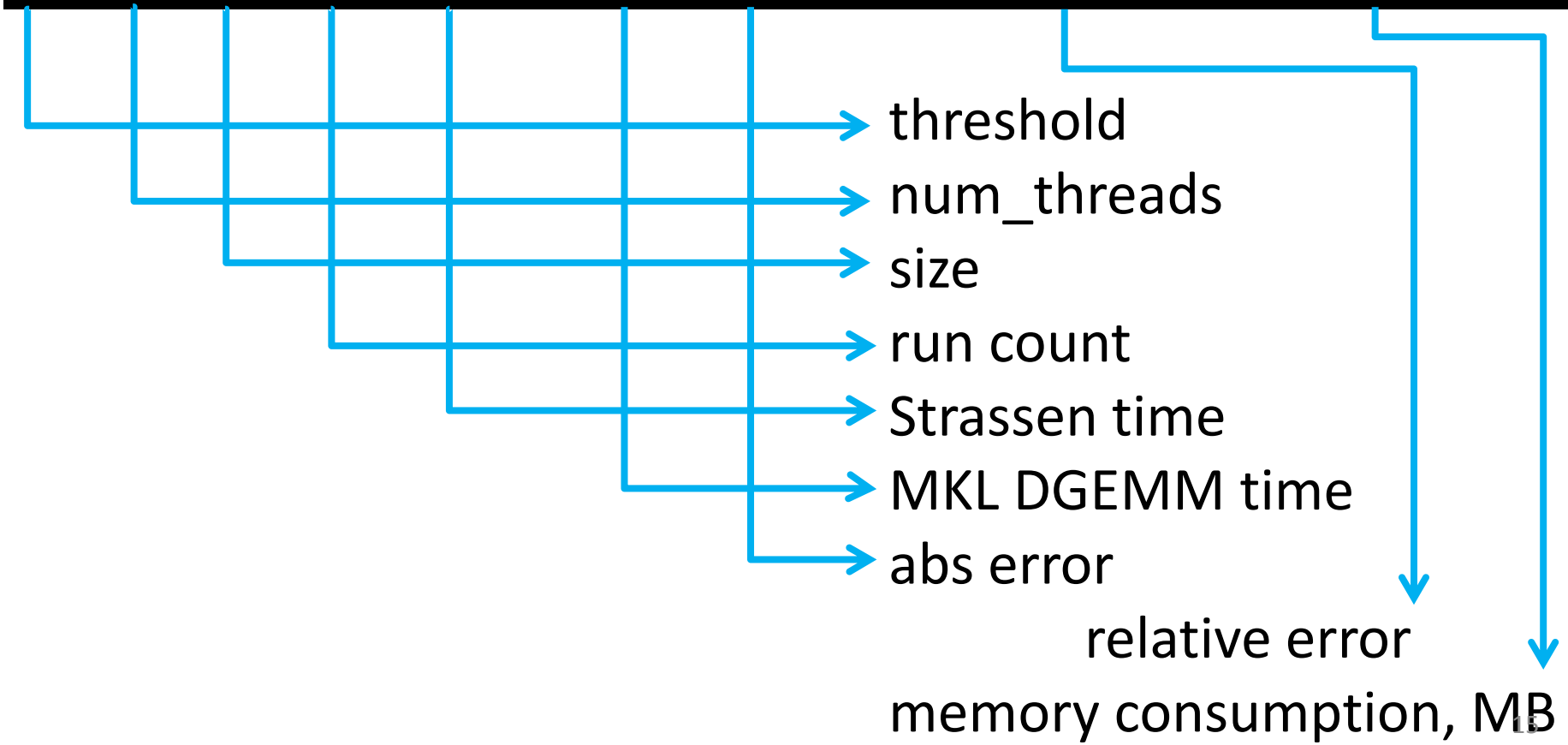
Build and run

```
./1/build_mic/run -t 512 -n 1 -s 4096 -r 1
./1/build_mic/run -t 256 -n 1 -s 4096 -r 1
./1/build_mic/run -t 1024 -n 1 -s 4096 -r 1
./1/build_mic/run -t 512 -n 60 -s 4096 -r 1
./1/build_mic/run -t 2048 -n 240 -s 8192 -r 1 -o
```

Max size ≈ 20000

Build and run

```
user1@ps-mic0:~/1/build_mic
$ ./1/build_mic/run -t 512 -n 60 -s 4096 -r 1
-sh: ./1/build_mic/run: No such file or directory
user1@ps-mic0:~/1/build_mic
$ ./run -t 512 -n 60 -s 4096 -r 1
512; 60; 4096; 1; 20.834; 0.534; 1.614353e-11; 1.544938e-14; 5.781524e+02;
```



Parallel Strassen algorithm

- Folder 2

```
CUnit        matrix_mult.c      matrix_strassen.h  run.c          unit_tests.c
extra_libs   matrix_mult.h      matrix_utils.c     test_utils.c   utils.c
Makefile     matrix_strassen.c  matrix_utils.h     test_utils.h   utils.h
```

Parallel Strassen algorithm

$$C = A * B = \begin{bmatrix} S_1 + S_2 + S_3 - S_4 & S_4 + S_6 \\ S_3 + S_5 & S_1 - S_5 + S_6 + S_7 \end{bmatrix}$$

$$S_1 = (A_{00} - A_{11}) * (B_{00} + B_{11})$$
$$S_2 = (A_{01} - A_{11}) * (B_{10} + B_{11})$$
$$S_3 = A_{11} * (B_{10} + B_{00})$$
$$S_4 = (A_{00} + A_{01}) * B_{11}$$
$$S_5 = (A_{10} + A_{11}) * B_{00}$$
$$S_6 = A_{00} * (B_{01} - B_{11})$$
$$S_7 = (A_{10} - A_{00}) * (B_{00} - B_{01})$$

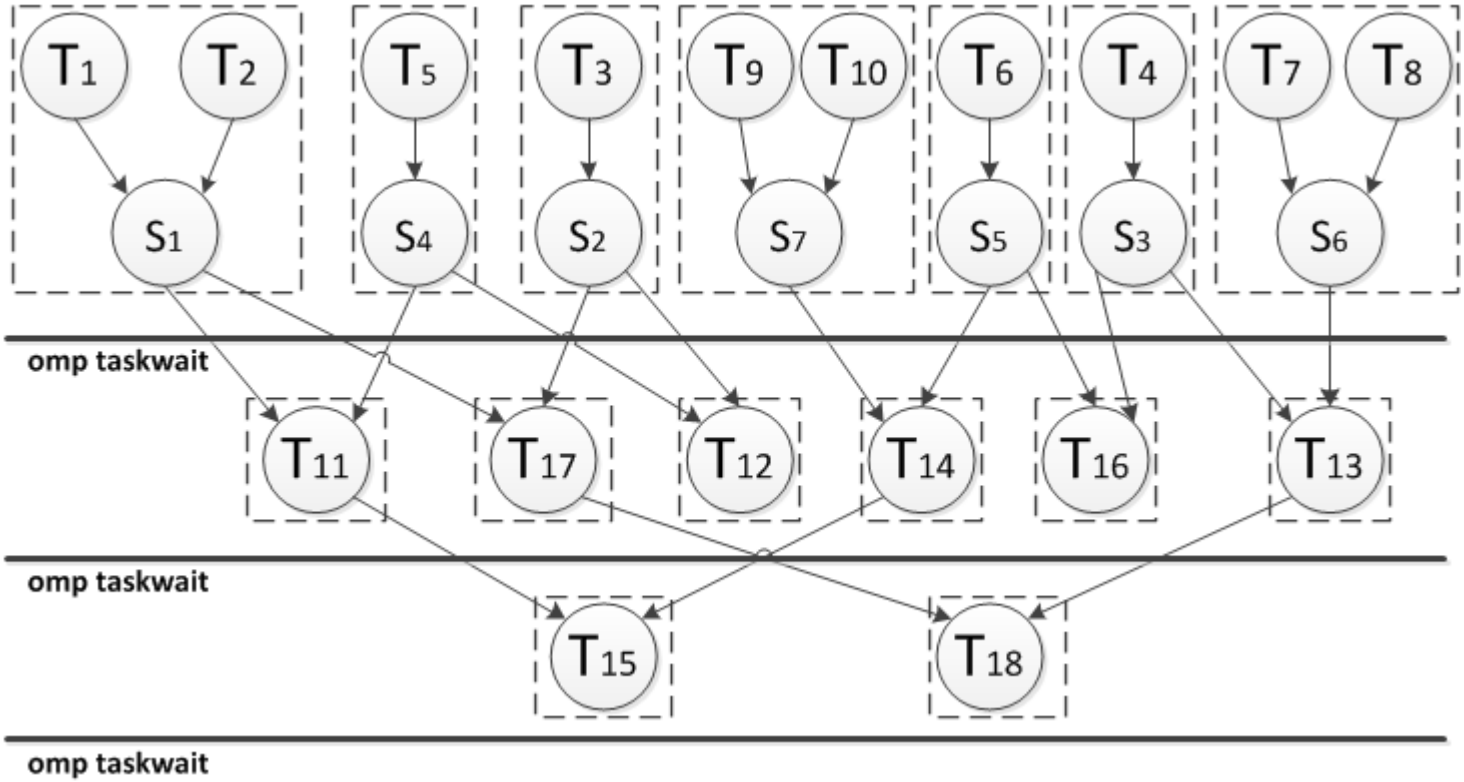
$$T_1 = A_{0,0} + A_{1,1}$$
$$T_2 = B_{0,0} + B_{1,1}$$
$$T_3 = A_{1,0} + A_{1,1}$$
$$T_4 = B_{0,1} - B_{1,1}$$
$$T_5 = B_{1,0} - B_{0,0}$$
$$T_6 = A_{0,0} + A_{0,1}$$
$$T_7 = A_{1,0} - A_{0,0}$$
$$T_8 = B_{0,0} + B_{0,1}$$
$$T_9 = A_{0,1} - A_{1,1}$$
$$T_{10} = B_{1,0} + B_{1,1}$$

$$S_1 = T_1 \cdot T_2$$
$$S_2 = T_3 \cdot B_{0,0}$$
$$S_3 = A_{0,0} \cdot T_4$$
$$S_4 = A_{1,1} \cdot T_5$$
$$S_5 = T_6 \cdot B_{1,1}$$
$$S_6 = T_7 \cdot T_8$$
$$S_7 = T_9 \cdot T_{10}$$

$$T_{11} = S_1 + S_4$$
$$T_{12} = S_2 + S_4$$
$$T_{13} = S_3 + S_6$$
$$T_{14} = S_7 - S_5$$
$$T_{16} = S_3 + S_5$$
$$T_{17} = S_1 - S_2$$

$$T_{15} = T_{11} + T_{14}$$
$$T_{18} = T_{13} + T_{17}$$

Task parallelism



Build and run

```
./build_x64.sh
./build_mic.sh

./run -t 512 -n 60 -s 8192 -r 1

./run -t 512 -n 120 -s 8192 -r 1

./run -t 512 -n 180 -s 8192 -r 1

./run -t 512 -n 240 -s 8192 -r 1
```

Intel(R) Math Kernel Library vs nested parallelism

If your application uses OpenMP threading, you may need to provide additional settings:

- Set the environment variable OMP_NESTED=true, or alternatively call omp_set_nested(1), to enable OpenMP nested parallelism.
- Set the environment variable MKL_DYNAMIC=false, or alternatively call mkl_set_dynamic(0), to prevent Intel MKL from dynamically reducing the number of threads in nested parallel regions.

https://software.intel.com/en-us/node/528380

https://software.intel.com/en-us/node/528547

mkl_set_num_threads

Intel MKL environment variable: MKL_NUM_THREADS

```
#include "mkl_service.h"

…

mkl_set_num_threads(4);

// Intel MKL uses up to 4 threads

my_compute_using_mkl();
```

mkl_set_num_threads_local

```
#include "mkl_service.h"
void my_compute( int nt )
{
        // save the Intel MKL number of threads
        int save = mkl_set_num_threads_local( nt );
// Intel MKL functions use up to nt threads on this thread
        my_compute_using_mkl();
        // restore the Intel MKL number of threads
        mkl_set_num_threads_local( save );
}
```

mkl_domain_set_num_threads

Intel MKL environment variable: MKL_DOMAIN_NUM_THREADS

```
#include "mkl_service.h"

mkl_domain_set_num_threads(4, MKL_DOMAIN_BLAS);

// Intel MKL BLAS functions use up to 4 threads

my_compute_using_mkl_blas();

// Intel MKL FFT functions use the default number of threads

my_compute_using_mkl_dft();
```

MKL_DOMAIN_ALL	All function domains
MKL_DOMAIN_BLAS	BLAS Routines
MKL_DOMAIN_FFT	non-cluster Fourier Transform Functions
MKL_DOMAIN_VML	Vector Mathematical Functions
MKL_DOMAIN_PARDISO	PARDISO

Nested parallelism

```
mkl_set_dynamic(0);
omp_set_nested(1);
set_num_threads(num_threads);
mkl_set_num_threads(mkl_num_threads);
strassen( … );
…
mkl_set_num_threads(num_threads * mkl_num_threads);
dgemm( … );
```

Parallel version with nested parallelism

- Folder 3

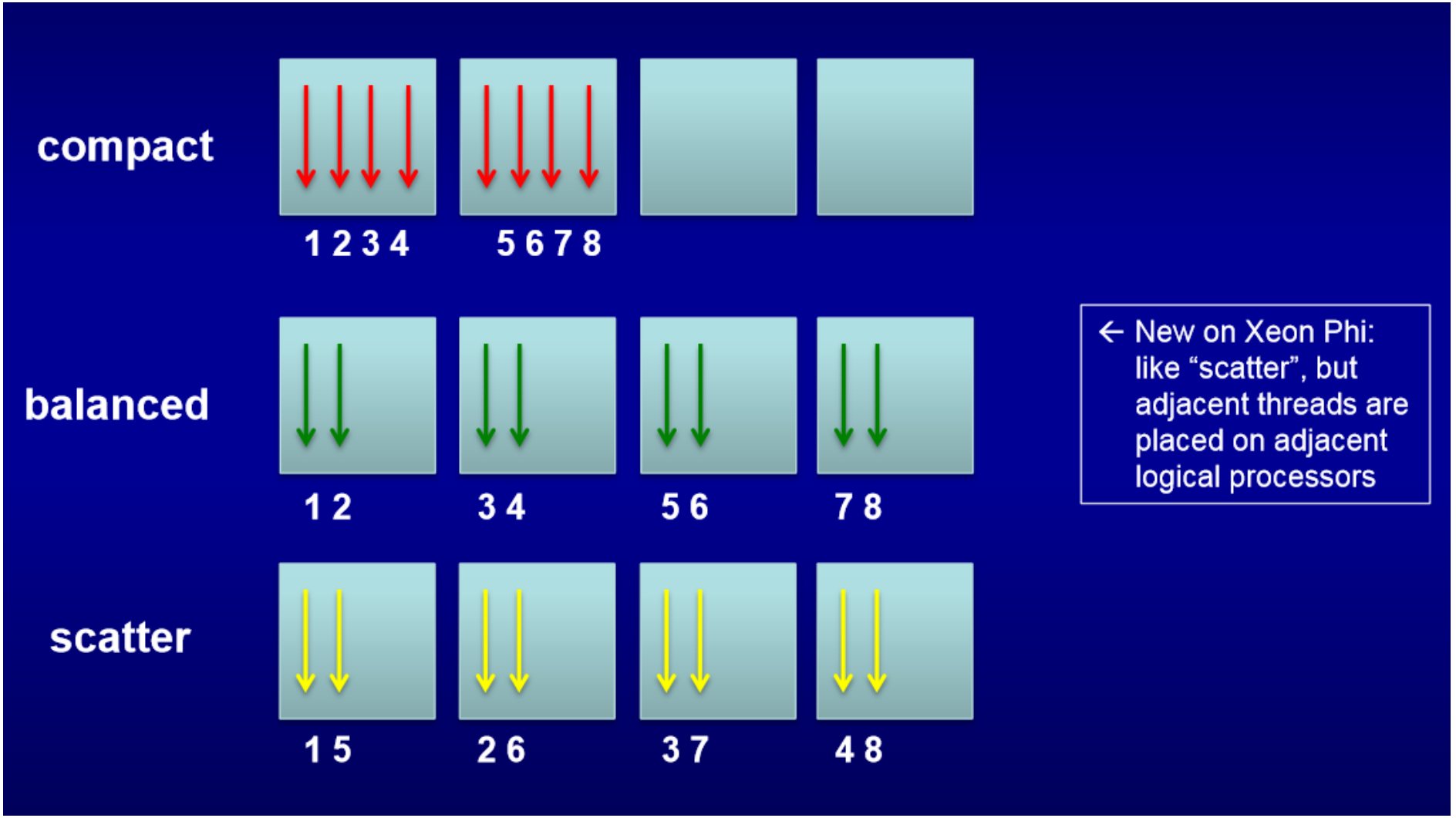
Build and run

```
./build_x64.sh
./build_mic.sh

./run -t 512 -n 60 -z 4 -s 8192 -r 1

./run -t 512 -n 120 -z 2 -s 8192 -r 1

./run -t 512 -n 180 -z 1 -s 8192 -r 1

./run -t 512 -n 240 -z 1 -s 8192 -r 1
```

OpenMP Thread Affinity Control

- https://software.intel.com/en-us/articles/openmp-thread-affinity-control
- Intel® Xeon Phi™ Coprocessor supports 4 thread contexts per core
- try different numbers of threads from $N-1$ threads to $4*(N-1)$ threads where N is the number of physical cores on the processor. Four simple experiments can be run: run the application with $(N-1)$ threads. Run with $2*(N-1)$, $3*(N-1)$ and $4*(N-1)$ to determine if the addition thread contexts give a performance benefit to your application.

KMP_AFFINITY distribution options



KMP_AFFINITY Examples

- `KMP_AFFINITY = granularity=fine,balanced`
- `KMP_AFFINITY = granularity=fine,compact`
- `KMP_AFFINITY = granularity=core,balanced`
- `KMP_AFFINITY=explicit,proclist=[1-240:1,0,241,242,243], granularity=fine`
- `KMP_AFFINITY=explicit,proclist=[0-243:2],granularity=fine`

Build and run

- `export KMP_AFFINITY=granularity=fine,scatter`
- `export KMP_AFFINITY=granularity=fine,balanced`
- `export KMP_AFFINITY=granularity=fine,compact`
- `export KMP_AFFINITY=granularity=core,balanced`
- `export KMP_AFFINITY=granularity=core,compact`

- `./run -t 512 -n 60 –z 4 -s 8192 -r 1`

KMP_PLACE_THREADS

value = (int ["C" | "T"] [delim] | delim) [int ["T"] [delim]] [int ["O"]];

Specifies the number of cores, with optional offset value and number of threads per core to use.

- "C" indicates Cores
- "T" indicates Threads
- "O" (letter O, not zero) is used to specify an Offset. Offset ignores granularity, Offset is the number of Cores to offset, starting from 0 "Core 0". Thus 1O would be the 2nd core in the package, aka "Core 1". Default is 0O
- Either cores or threads should be specified. If omitted, the default value is the available number of cores (threads).

KMP_PLACE_THREADS Examples

- 5C,3T,1O - use 5 cores with offset 1, 3 threads per core
- 5,3,1 - same as above
- 24 - use first 24 cores, 4 threads per core
- 2T - use all cores, 2 threads per core
- ,2 - same as above
- 3x2 - use 3 cores, 2 threads per core
- 4C,12O - use 4 cores with offset 12, all available threads per core

Extra tips

- Building Affinity Into an Executable
- Performance Tip: KMP_BLOCKTIME Parameter
- Performance Tip: -qopt-threads-per-core=1/2/3/4 (default is 4), gives a hint to the compiler about how many threads are likely to be running on the core for the application.
- Prefetch: https://software.intel.com/sites/default/files/managed/21/ea/5.3-prefetching-on-mic-5.pdf

Thank you for your attention

- singularis-lab.com

SINGULARIS LAB

software development