Intel Libraries

Intel MKL, Intel DAAL, and Intel TBB

Intel High Performance and Throughput Computing (EMEA)
Hans Pabst, July 9th 2015

Motivation

How and where to optimize?

1. Appropriate algorithm
2. **Performance Library**
3. Multicore
4. SIMD

```
for (int i = 0; i < M; ++i) {
 for (int j = 0; j < N; ++j) {
  c[i*K+j] = 0;
  for (int k = 0; k < K; ++k) {
   c[i*K+j] += a[i*N+k]
             * b[k*K+j];
  }
 }
}
```

Delivered Values

- Easy access to high perf.
- Rich functionality
- Support

Performance Library

Intel® Math Kernel Library

Intel® MKL

Intel® Math Kernel Library (Intel® MKL)

Linear Algebra
• BLAS, Sparse BLAS • LAPACK solvers • Sparse Solvers (DSS, PARADISO) • Iterative solver (RCI) • ScaLAPACK, PBLAS

Fast Fourier Transforms
• Multidimensional • FFTW interfaces • Cluster FFT • Trig. Transforms • Poisson solver • Convolution via VSL

Vector Math
• Trigonometric • Hyperbolic • Exponential, Logarithmic • Power / Root

Random Number Gen.
• Congruential • Wichmann-Hill • Mersenne Twister • Sobol • Neiderreiter • Non-deterministic

Summary Statistics
• Kurtosis • Variation coefficient • Quantiles • Order statistics • Min/max • Variance-covariance

Data Fitting
• Spline-based • Interpolation • Cell search

Intel® MKL: What's New in Version 11.x?

Release Notes (good source of what's new)

https://software.intel.com/en-us/articles/intel-mkl-113-release-notes

https://software.intel.com/en-us/articles/intel-mkl-112-release-notes

https://software.intel.com/en-us/articles/intel-mkl-111-release-notes

https://software.intel.com/en-us/articles/intel-mkl-110-release-notes

Selection of What's New

Optimizations for latest ISAs and extensions: Intel® Xeon Phi™ Coprocessor (native/offload, and auto-offload), Intel® AVX2, Intel® AVX-512, etc.

Conditional Numerical Reproducibility (CNR), Verbose Mode, Small Matrix Multiplication "inlining", Cluster Sparse Solver (PARDISO)

Composability with TBB, new/complementary handle based SpBLAS API, additional RNG algorithms, new C-only documentation, etc.

Intel® MKL: Documentation

Getting Started

https://software.intel.com/en-us/articles/intel-mkl-113-getting-started

Reference Manual

https://software.intel.com/en-us/mkl_11.2_ref

User's Guide

https://software.intel.com/en-us/mkl_11.2_ug_lin

https://software.intel.com/en-us/mkl_11.2_ug_win

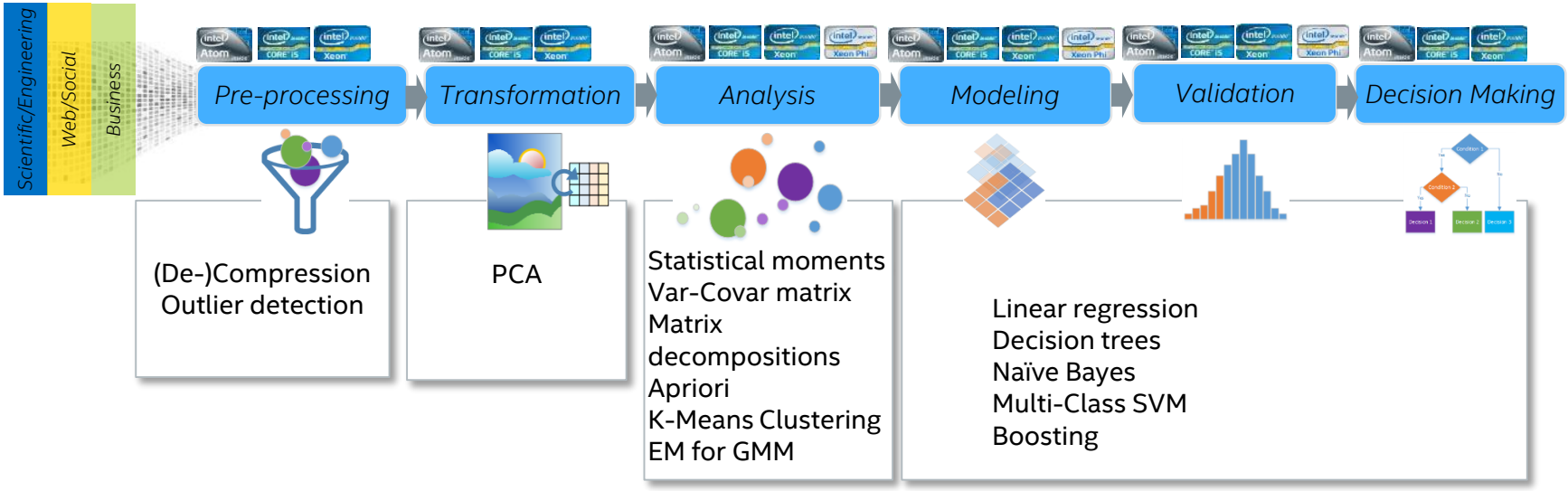
https://software.intel.com/en-us/mkl_11.2_ug_osx

Intel® Data Analytics Acceleration Library

Intel® DAAL

Intel® Data Analytics Acceleration Library

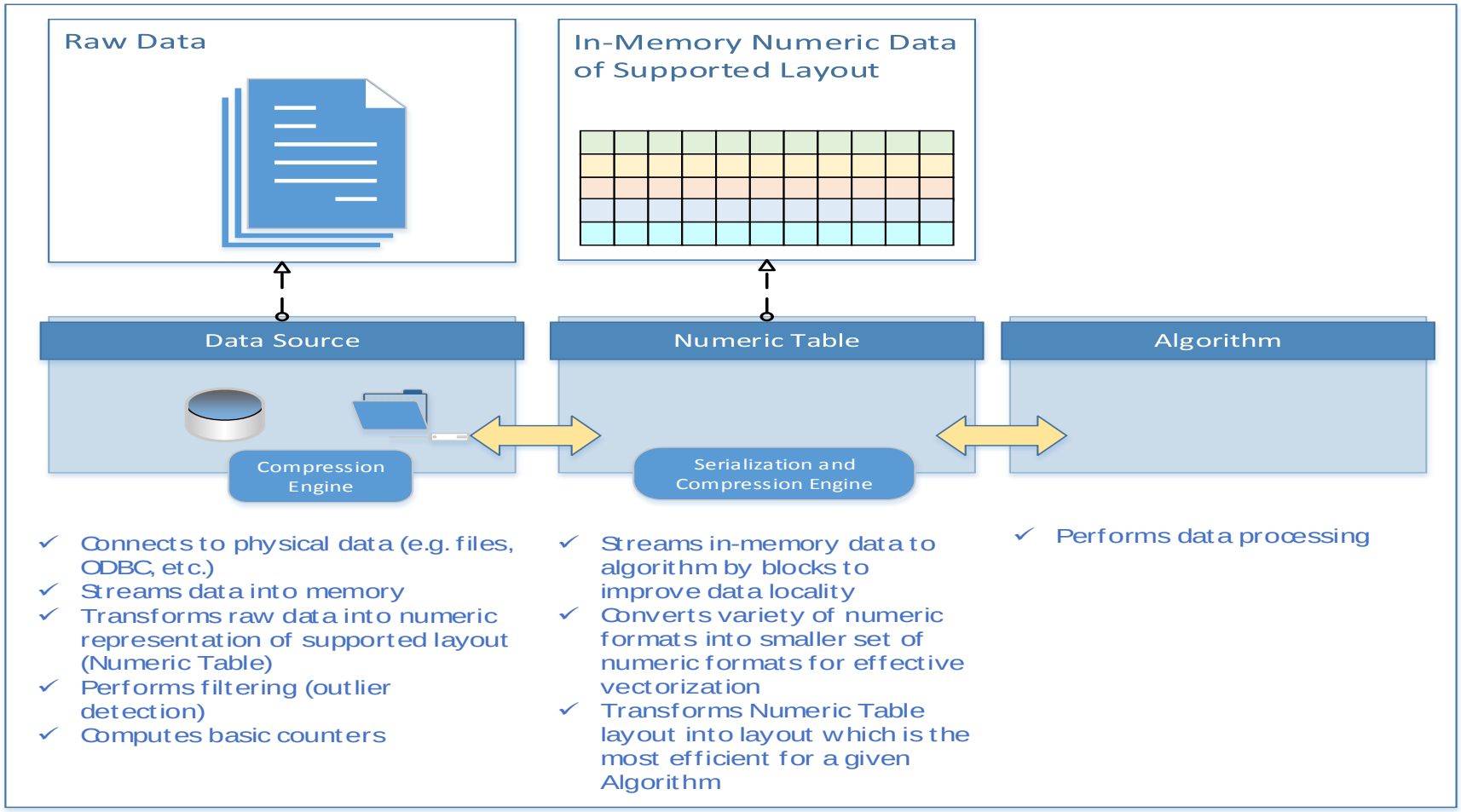
An industry leading end-to-end IA-based data analytics acceleration library of fundamental algorithms covering all data analysis stages.



Intel DAAL: What Is and Isn't?

It is …	It is not …
a performance library with C++ and Java APIs optimized for Intel architectures.	a programming environment or a cluster computing framework (like MATLAB, R, or Hadoop).
a collection of common building blocks for constructing high-end solutions in all stages of a data analytics project.	a black-box solution to tackle domain specific analytics needs.
abstracted from communication layers and data sources, to be easily integrated into different analytics platforms.	a toolkit or plug-in tied to a particular big data platform.
boosting performance of critical algorithms hence reducing time-to-value of your big data projects.	promoting fancy algorithms as the silver bullet for all you big data needs.

Intel DAAL: Typical Work Flow



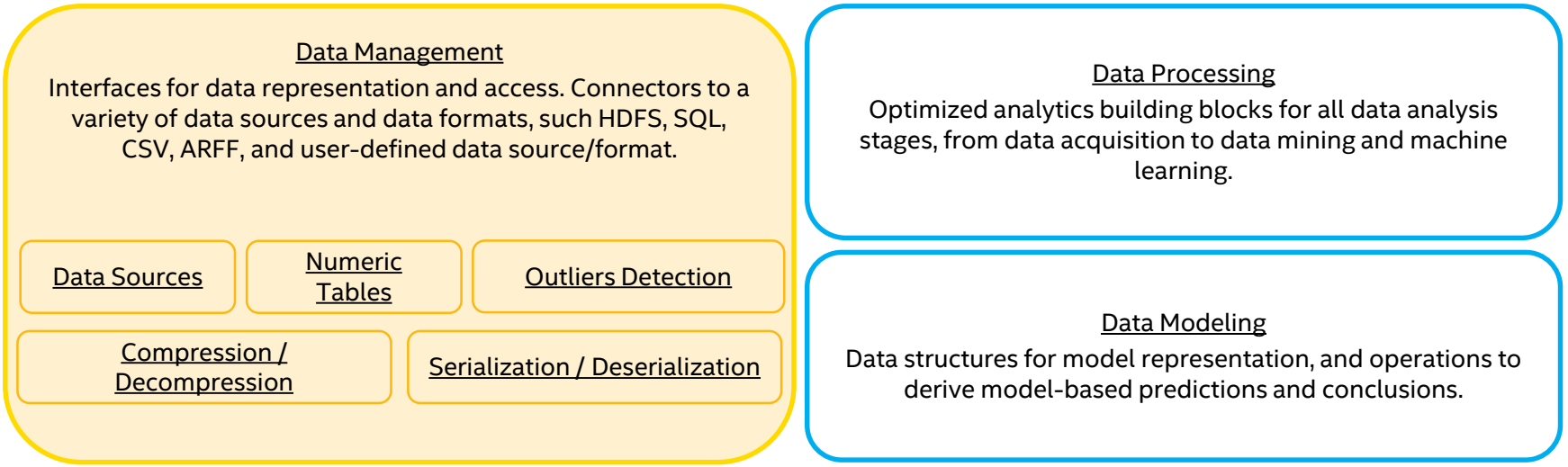
- ✓ Connects to physical data (e.g. files, ODBC, etc.)
- ✓ Streams data into memory
- ✓ Transforms raw data into numeric representation of supported layout (Numeric Table)
- ✓ Performs filtering (outlier detection)
- ✓ Computes basic counters

- ✓ Streams in-memory data to algorithm by blocks to improve data locality
- ✓ Converts variety of numeric formats into smaller set of numeric formats for effective vectorization
- ✓ Transforms Numeric Table layout into layout which is the most efficient for a given Algorithm

- ✓ Performs data processing

Intel DAAL: Data Management and Data Processing

Tasks: Raw data acquisition, filtering, conversion.

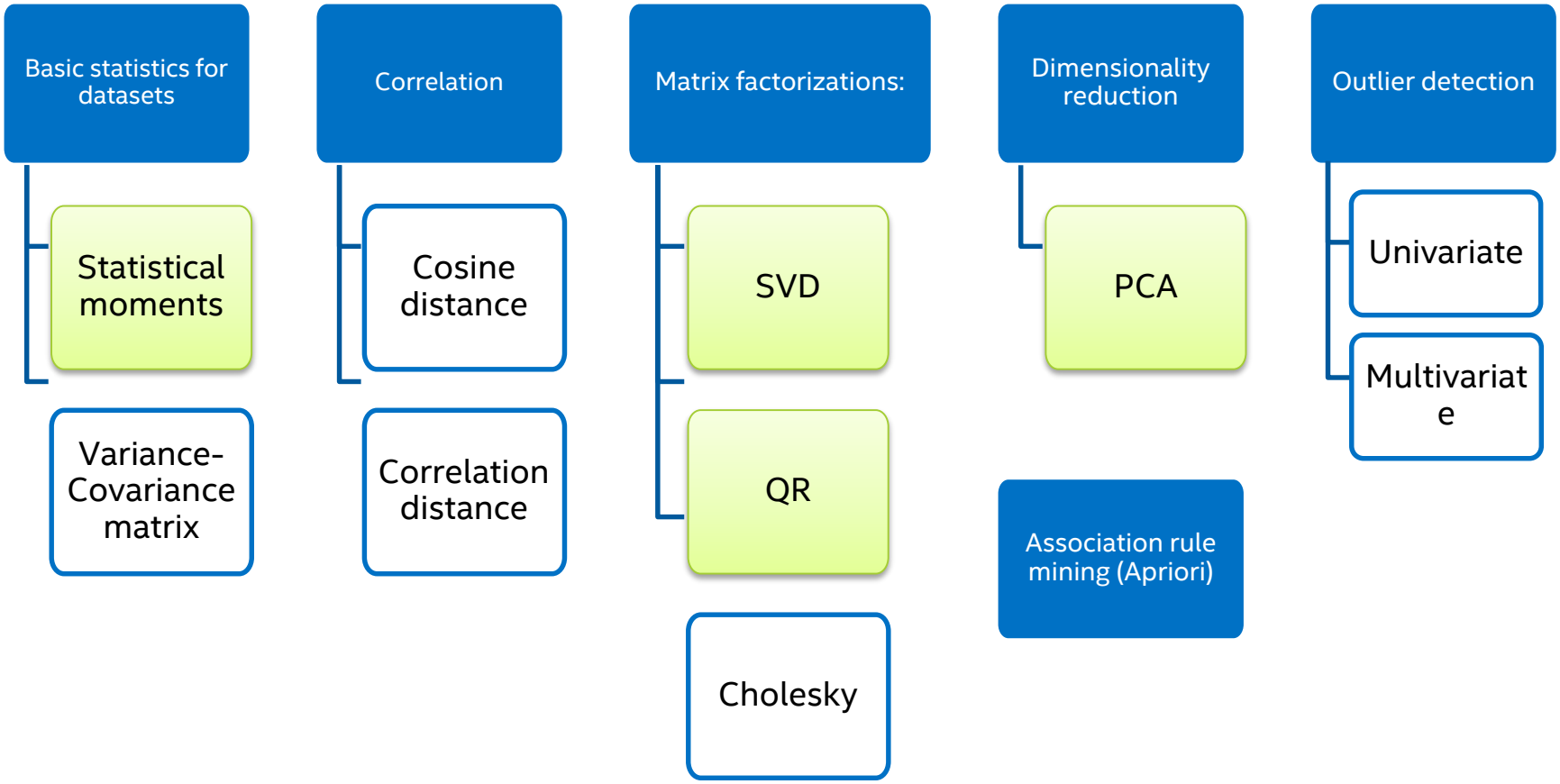


Algorithms for data mining and machine learning.

Computation modes: batch, distributed, streaming.

Common tasks: compute, merge, finalizeMerge, finalizeStream.

Intel DAAL: Algorithms for Data Transformation and Analysis



Algorithms support streaming and distributed processing in the current release.

Intel® Threading Building Blocks

Intel® TBB

Intel® Threading Building Blocks (Intel® TBB)

What

- Widely used C++ template library for task parallelism.
- Features
- Parallel algorithms and data structures.
- Threads and synchronization primitives.
- Scalable memory allocation and task scheduling.

Also available as open source at threadingbuildingblocks.org

https://software.intel.com/intel-tbb

Benefit

- Rich feature set for general purpose parallelism.
- Available as an open source and a commercial license.
- Supports C++, Windows*, Linux*, OS X*, other OS's.
- Commercial support for Intel® Atom™, Core™, Xeon® processors, and for Intel® Xeon Phi™ coprocessors

Simplify Parallelism with a Scalable Parallel Model

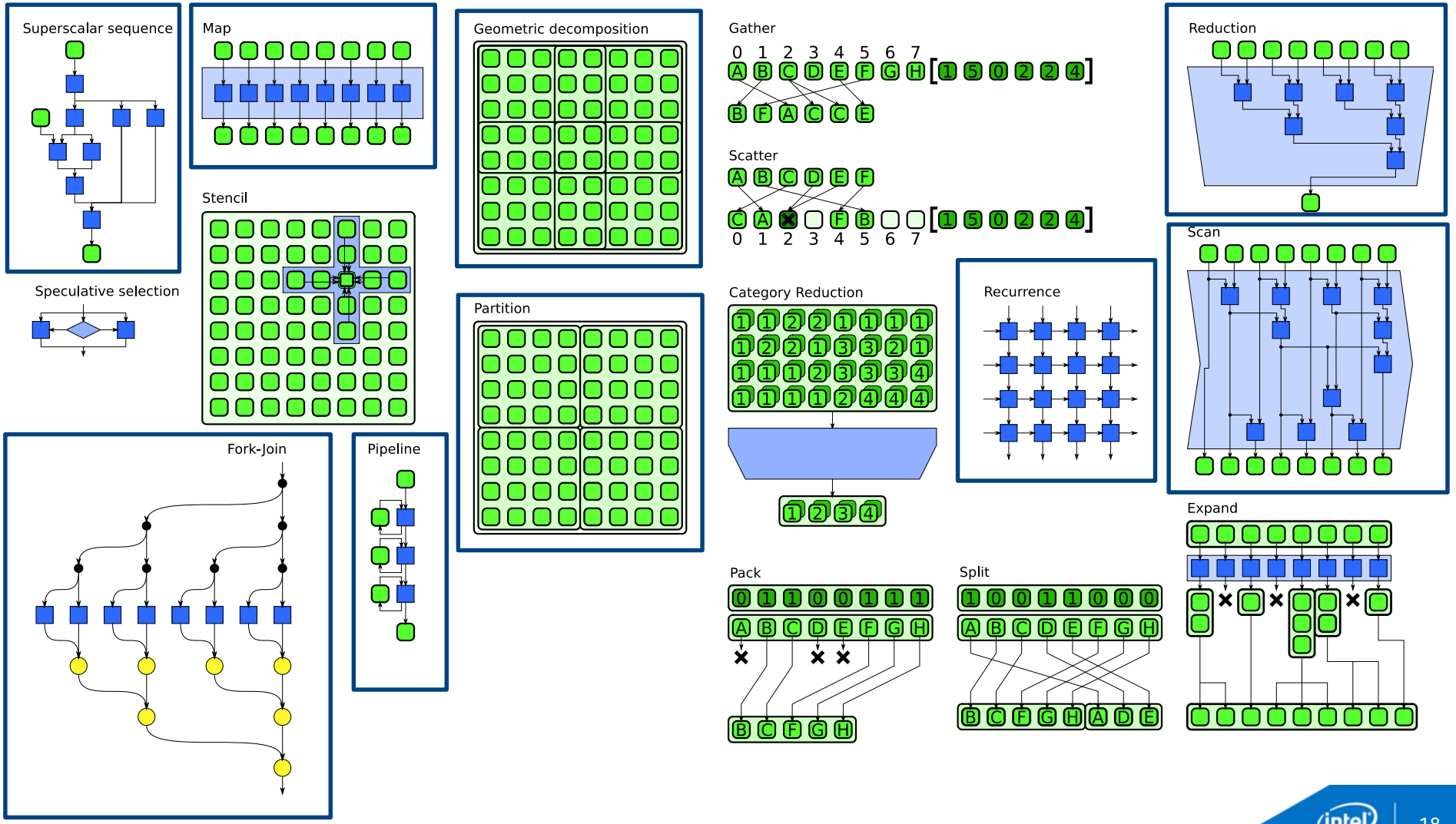
Design patterns

- Parallel pattern: commonly occurring combination of task distribution and data access
- A small number of patterns can support a wide range of applications

→ **Identify and use parallel patterns**
Examples: reduction, or pipeline

→ **TBB has primitives and algorithms for most common patterns** – don't reinvent a wheel

Parallel Patterns



Parallel algorithms and data structures

Threads and synchronization

Memory allocation and task scheduling

Rich Feature Set
for Parallelism

Generic Parallel Algorithms

Efficient scalable way to exploit the power of multi-core without having to start from scratch.

Flow Graph

A set of classes to express parallelism as a graph of compute dependencies and/or data flow

Concurrent Containers

Concurrent access, and a scalable alternative to containers that are externally locked for thread-safety

Synchronization Primitives

Atomic operations, a variety of mutexes with different properties, condition variables

Task Scheduler

Sophisticated work scheduling engine that empowers parallel algorithms and the flow graph

Timers and Exceptions

Thread-safe timers and exception classes

Threads

OS API wrappers

Thread Local Storage

Efficient implementation for unlimited number of thread-local variables

Memory Allocation

Scalable memory manager and false-sharing free allocators

Features and Functions List

Parallel algorithms and data structures

Threads and synchronization

Memory allocation and task scheduling

Generic Parallel Algorithms

- parallel_for
- parallel_reduce
- parallel_for_each
- parallel_do
- parallel_invoke
- parallel_sort
- parallel_deterministic_reduce
- parallel_scan
- parallel_pipeline
- pipeline

Flow Graph

- graph
- continue_node
- source_node
- function_node
- multifunction_node
- overwrite_node
- write_once_node
- limiter_node
- buffer_node
- queue_node
- priority_queue_node
- sequencer_node
- broadcast_node
- join_node
- split_node
- indexer_node

Concurrent Containers

- concurrent_unordered_map
- concurrent_unordered_multimap
- concurrent_unordered_set
- concurrent_unordered_multiset
- concurrent_hash_map
- concurrent_queue
- concurrent_bounded_queue
- concurrent_priority_queue
- concurrent_vector
- concurrent_lru_cache

Synchronization Primitives

- atomic
- mutex
- recursive_mutex
- spin_mutex
- spin_rw_mutex
- speculative_spin_mutex
- speculative_spin_rw_mutex
- queuing_mutex
- queuing_rw_mutex
- null_mutex
- null_rw_mutex
- reader_writer_lock
- critical_section
- condition_variable
- aggregator (preview)

Task Scheduler

- task
- task_group
- structured_task_group
- task_group_context
- task_scheduler_init
- task_scheduler_observer
- task_arena

Exceptions

- tbb_exception
- captured_exception
- movable_exception

Threads & timers

Thread

tick_count

Thread Local Storage

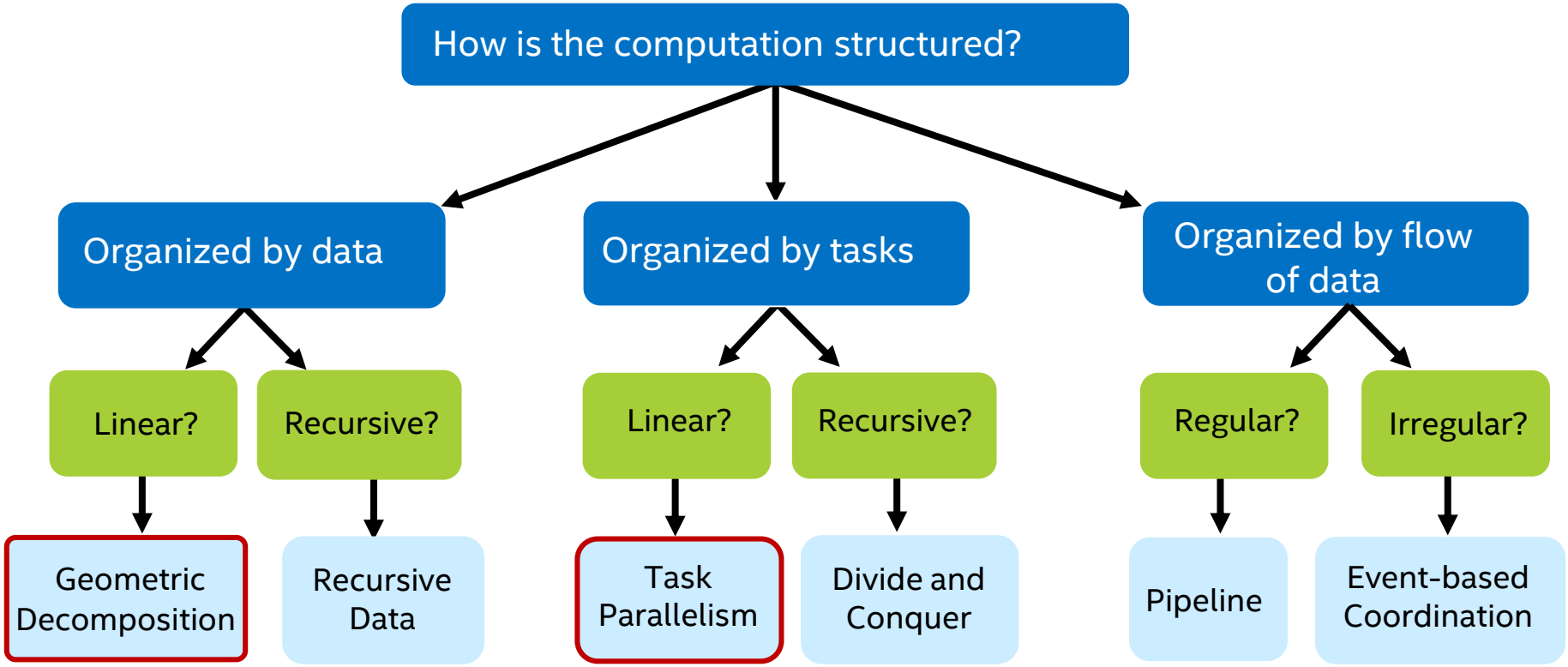
- combinable
- enumerable_thread_specific

Memory Allocation

- tbb_allocator
- scalable_allocator
- cache_aligned_allocator
- zero_allocator
- aligned_space
- memory_pool (preview)

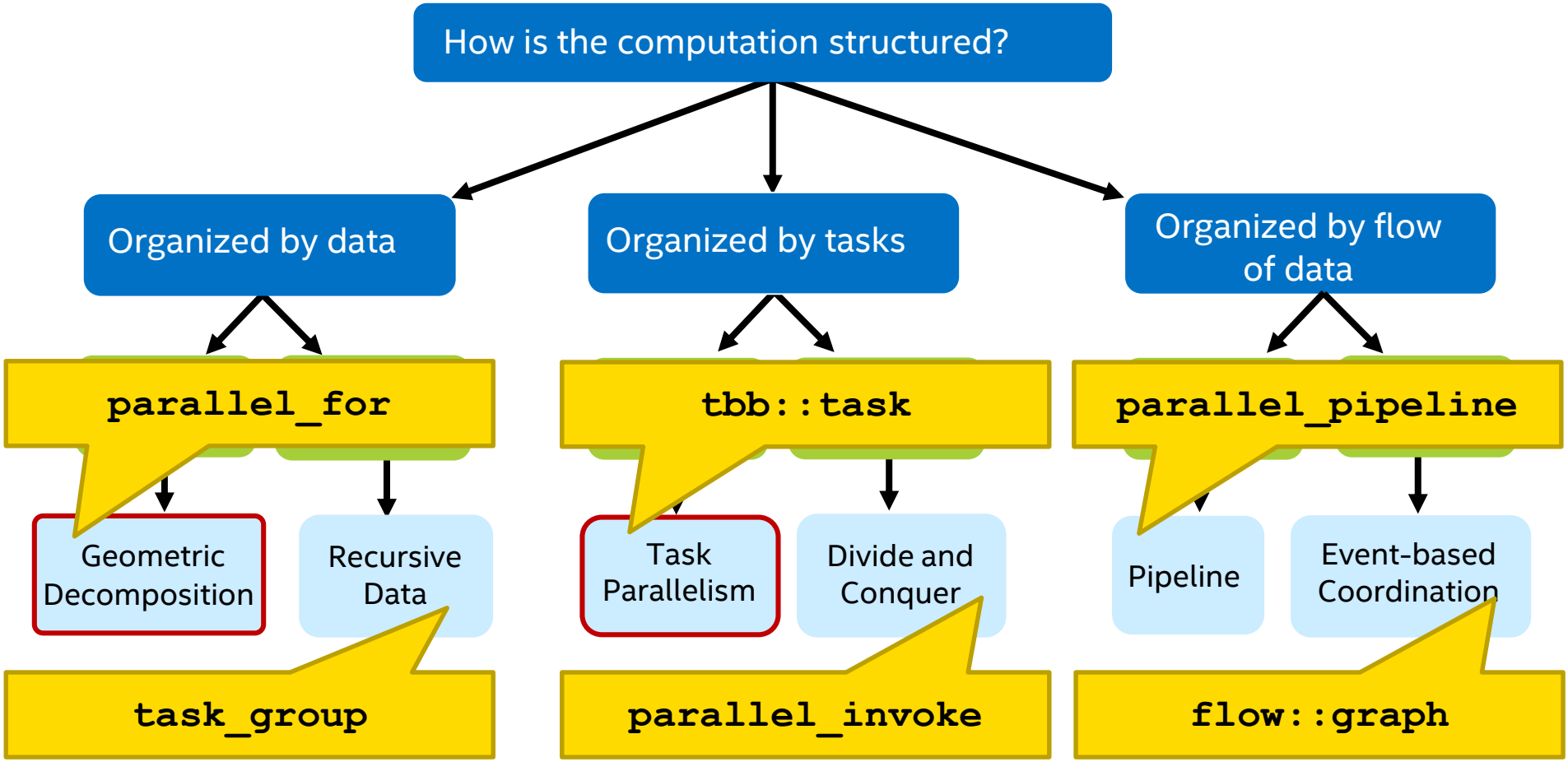
Algorithm Structure Design Space

Structure used to organize parallel computations



Algorithm Structure Design Space

Structure used to organize parallel computations



Legal Disclaimer & Optimization Notice

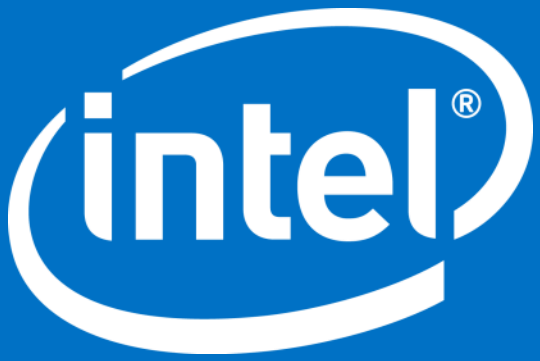
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING 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.

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


intel®

Backup

Intel MKL Code Samples

Intel® MKL: Intel® Compiler Math Library

Intel Compiler: scalar/vector math fn., and pseudo Intrinsics

- **IMF**: Intel Math Functions (scalar)

  -fimf-precision=<high|medium|low>

  Default/usually: scalar/high and vect./medium
  high: ~0.55 ulps, medium: ~2 ulps (but < 4 ulps)

  -fimf-arch-consistency=<true|false>

  Default is "false" even with –fp-model=precise

  Not available across 32-bit / 64-bit

- **SVML**: Short Vector Math Functions (pseudo Intrinsics)

  General form: _mm*_svml_[function]_p[s|d]

  e.g., _mm_svml_round_ps, or _mm256_erfc_pd

Intel® MKL: vectorized and parallelized math functions

- **VML**: optimized for throughput – three accuracy/performance levels

Intel® MKL: Vector Math Library (VML)

High Accuracy (HA)

- Correct rounding (>99%)
- Behaves according to C99
- Slowest, default mode

Low Accuracy (LA)

- At most 2 lsb incorrect
- Behaves according to C99
- **30-50% faster than HA**

Enhanced Performance (EP)

- ~1/2 incorrect bits
- **30-50% faster than LA**

```
#include <mkl_vml.h>

int main()
{
  double  in[1000];
  double out[1000];
  vmlSetMode(VML_EP)
  vdExp(1000, in, out);
}
```

* http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/vml/functions/_performanceall.html
http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/vml/functions/_accuracyall.html
http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/vml/functions/exp.html

Intel® MKL: SGEMM (CBLAS)

```
using namespace std;

vector<float> a(arows * acols);
vector<float> b(acols * bcols);
vector<float> c(arows * bcols);
const float alpha = 1, beta = 0;

transform(a.begin(), a.end(), a.begin(), [](float /*dummy*/)
  { return static_cast<float>(rand()); });
transform(b.begin(), b.end(), b.begin(), [](float /*dummy*/)
  { return static_cast<float>(rand()); });
transform(c.begin(), c.end(), c.begin(), [](float /*dummy*/)
  { return static_cast<float>(rand()); });

cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
  arows, bcols, acols, alpha, &a[0], acols, &b[0],
  bcols, beta, &c[0], bcols);
```

* No overloaded functions (C interface). Note, CBLAS vs. BLAS is to get row- vs. col-major storage.

Intel® MKL: Setting Affinity (OpenMP)*

```
[e5-2670] $ source /opt/intel/composerxe/bin/compilervars.sh intel64

[e5-2670] $ icc -O2 -mkl dgemm.c -o dgemm
[e5-2670] $ env KMP_AFFINITY=compact,1 ./dgemm
```

Intel® MKL: C++ Math Libraries (Wrapper)

Several C++ template libraries available*

- **Armadillo**, **Eigen**, etc.

Typical criterions when deciding

- Use of expression templates to enable lazy evaluation and to avoid intermediate temporaries
- Data containers able to allocate aligned buffers and able to wrap existing memory layouts (user-allocated)
- Simple configuration (preprocessor symbols preferred) and compiler-agnostic (OS portable)

* http://software.intel.com/en-us/articles/intelr-mkl-and-c-template-libraries

Intel® MKL: C++ Wrapper Code

```
template<typename T, typename U> void gemm(T* result, const T* a, const T* b,
  U arows, U acols, U bcols, T alpha = 1, T beta = 0)
{
  struct local {
    const char atrans = 'T', btrans = 'T';

    static void gemm(float* result, const float* a, const float* b,
      MKL_INT arows, MKL_INT acols, MKL_INT bcols, float alpha, float beta)
    {
      sgemm(&atrans, &btrans, &arows, &bcols, &acols, &alpha, a, &acols, b,
        &bcols, &beta, result, &bcols);
    }

    static void gemm(double* result, const double* a, const double* b,
      MKL_INT arows, MKL_INT acols, MKL_INT bcols, double alpha, double beta)
    {
      dgemm(&atrans, &btrans, &arows, &bcols, &acols, &alpha, a, &acols, b,
        &bcols, &beta, result, &bcols);
    }
  };

  local::gemm(result, a, b,
    static_cast<MKL_INT>(arows),
    static_cast<MKL_INT>(acols),
    static_cast<MKL_INT>(bcols),
    alpha, beta);
}
```

* Note, the Intel MKL C/BLAS interfaces are const-correct. Further, MKL_INT depends on LP64 vs. ILP64.

Intel® MKL: Vector Statistics Library (VSL)

```
void process_signal_mkl(size_t size,
  const float xin[], const float yin[],
  float       xout[], float       yout[])
{
  static MKL_INT n = static_cast<MKL_INT>(size);
  static VSLCorrTaskPtr task = 0;

  if (n != size || 0 == task) {
    vslCorrDeleteTask(&task);
    mkl_size = static_cast<MKL_INT>(size);
    vslsCorrNewTask1D(&task, VSL_CORR_MODE_AUTO,
                      n, n, n);
  }
  std::copy(xin, xin + size, xout);
  vslsCorrExec1D(task, yin, 1, yin, 1, yout, 1);
}
```