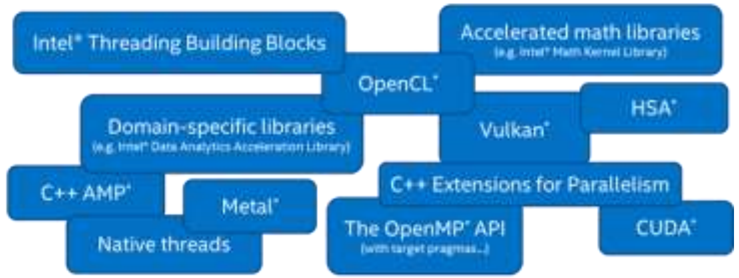
SOLVING PERFORMANCE PUZZLE BY USING INTEL® THREADING BUILDING BLOCKS

Evgeny Fiksman
Intel Corporation
October 2017

CHALLENGES IN PROGRAMING OF MODERN SYSTEMS

intel
Intel® Xeon® Bronze Processor
Intel® Xeon® Silver Processor
Intel® Xeon® Gold Processor
Intel® Xeon® Platinum Processor
Xeon Phi™ Processor

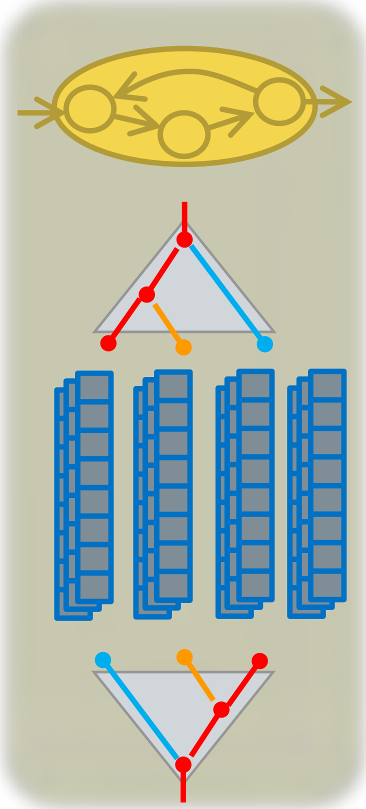
+


Intel® Threading Building Blocks
Accelerated math libraries (e.g. Intel® Math Kernel Library)
OpenCL*
Domain-specific libraries (e.g. Intel® Data Analytics Acceleration Library)
HSA*
Vulkan*
C++ AMP*
Metal*
C++ Extensions for Parallelism
Native threads
The OpenMP* API (with target pragmas...)
CUDA*

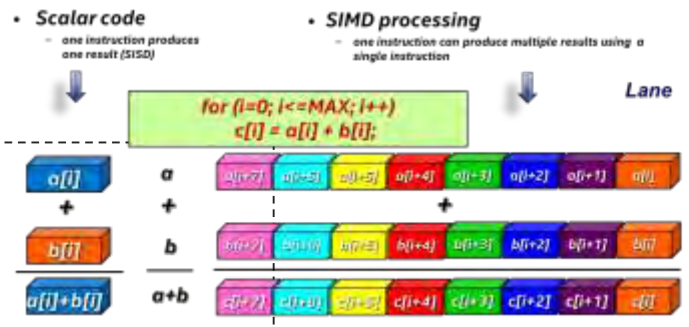
- Hardware Progress in 6 years
  - Over 3x more cores, 8 (Intel® Xeon E5) vs 28 (Intel® Xeon Platinum 8180) cores, or 72 (Intel® Xeon Phi™ 2nd gen)
  - 4x wider registers , Intel® SSE4.2 vs Intel® AVX-512
  - Under same power envelope -> Lower frequency
  - Heterogeneity

THE THREE LEVELS OF PARALLELISM

- Task / Message Driven
  - GRID engines, MPI, Intel TBB Flow Graph
- Fork-Join Parallel Regions
  - OpenMP*
  - Intel TBB
  - Parallel STL in C++17(Requires runtime support)
- (auto-)Vectorization / SIMD
  - Libraries
  - Intrinsics
  - OpenMP 4.x(Requires compiler support)
  - Parallel STL in C++17(Requires compiler support)



VECTORIZATION



Scalar code
one instruction produces one result (SISD)
SIMD processing
one instruction can produce multiple results using a single instruction
Lane
for (i=0; i<=MAX; i++)
c[i] = a[i] + b[i];
a[i]
+
b[i]
a[i]+b[i]
a
+
b
a+b
a[i+7] a[i+6] a[i+5] a[i+4] a[i+3] a[i+2] a[i+1] a[i]
+
b[i+7] b[i+6] b[i+5] b[i+4] b[i+3] b[i+2] b[i+1] b[i]
c[i+7] c[i+6] c[i+5] c[i+4] c[i+3] c[i+2] c[i+1] c[i]

```
void foo(const double* a, const double* b, double* c, int n)
{
    for(int i=0; i<n; ++i)
        c[i] = bar(a[i],b[i]);
}
```

“Serial brick”

```
void foo(const double* a, const double* b, double* c, int n)
{
    #pragma omp simd
    for(int i=0; i<n; ++i)
        c[i] = bar(a[i],b[i]);
}
```

“vectorization”

```
#pragma omp declare simd
double bar(double a, double b)
   { return sqrt(a/b); }
…
void foo(const double* a, const double* b, double* c, int n)
{
    #pragma omp simd
    for(int i=0; i<n; ++i)
        c[i] = bar(a[i], b[i]);
}
```

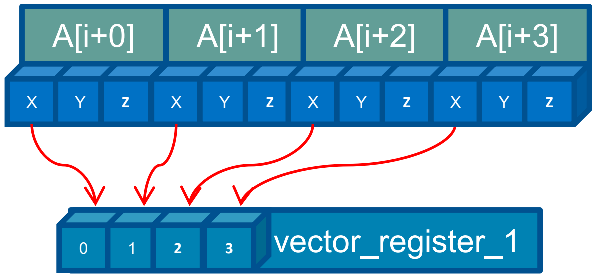
Vector function

MEMORY LAYOUT OPTIMIZATION – INTEL® SDLT

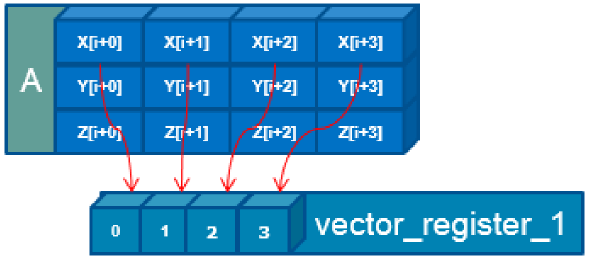
SIMD DATA LAYOUT TEMPLATES

- A C++11 template library providing concepts of Containers, Accessors and Offsets
  - Containers encapsulate the in memory data layout of an Array of "Plain Old Data" objects.
  - SIMD loops use accessors with an array subscript operator (just like C++ arrays) to read from or write to the objects in the Containers.
  - Offsets can be embedded in accessors or applied to a Index passed to the accessors array subscript operator.
- Usages:
  - Image processing
  - 2D PDE solvers
  - Interpolators

AOS in Memory Layout



SOA in Memory Layout


A
X[i+0] X[i+1] X[i+2] X[i+3]
Y[i+0] Y[i+1] Y[i+2] Y[i+3]
Z[i+0] Z[i+1] Z[i+2] Z[i+3]
0 1 2 3
vector_register_1

INTEL SDLT EXAMPLE

```
using namespace sdlt;
struct YourStruct
{
    float x;
    float y;
    float z;
};
SDLT_PRIMITIVE(YourStruct, x, y, z)
…
auto shape = n_extents[128][256][512];
typedef n_container<YourStruct, layout::soa, decltype(shape)> Container3d;

Container3d input(shape), output(shape);
auto inputs  = input.const_access();
auto outputs = output.access();

for(int z = 0; z < shape.extent_d0(); ++z) {
    for(int y = 0; y < shape.extent_d1(); ++y) {
        #pragma omp simd
        for(int x = 0; x < shape.extent_d2(); ++x) {
            YourStruct val = inputs[z][y][x];
            YourStruct result = … // compute result
            outputs[z][y][x] = result;
}}}
```

C++ lacks compile time reflection, so the user must provide SDLT with some information on the layout of YourStruct.

FORK-JOIN: INTEL® THREADING BUILDING BLOCKS (INTEL® TBB)

- A widely used C++ template library for parallel programming
- What
  - Task based scheduling
  - Parallel algorithms and data structures
    - parallel_for, parallel_reduce, ….
    - concurrent_hash, concurrent_queue, …
  - Dependency graphs and data flow primitives
  - Thread and synchronization primitives
    - atomics, TLS, mutex, …
  - Scalable memory allocation – efficient also for non threaded applications
- Benefits
  - Is a library-only solution that does not depend on compiler support
  - Is both a commercial product and an open-source(Apache v2.0)
  - Supports C++, Windows*, Linux*, OS X*, Android* and other OSes
  - Commercial support for Intel® Atom™, Core™, Intel® Xeon® & Intel® Xeon Phi™ processors

FORK-JOIN:PARALLEL REGIONS

```
#pragma omp declare simd
double bar(double a, double b)
   { return sqrt(a/b); }
void foo(const double* a, const double* b, double* c, int n)
{
    #pragma omp simd
    for(int i=0; i<n; ++i)
        c[i] = bar(a[i], b[i]);
}
```

“vectorized brick”

```
#pragma omp declare simd
double bar(double a, double b)
   { return sqrt(a/b); }

tbb::parallel_for(tbb::blocked_range<int>(0,n,STEP),
    [&](const tbb::blocked_range<in>& r)
    {
        auto begin = r.begin(), end = r.end();
        #pragma omp simd
        for(auto ii=begin; ii<end; ++ii)
            c[ii] = bar(a[ii], b[ii]);
    }
);
```

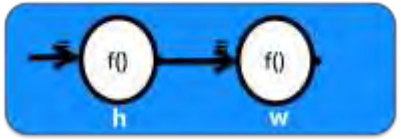
C++11: Lambda Expression

OpenMP*4.0 simd directive co-exists with a different multi-threading framework

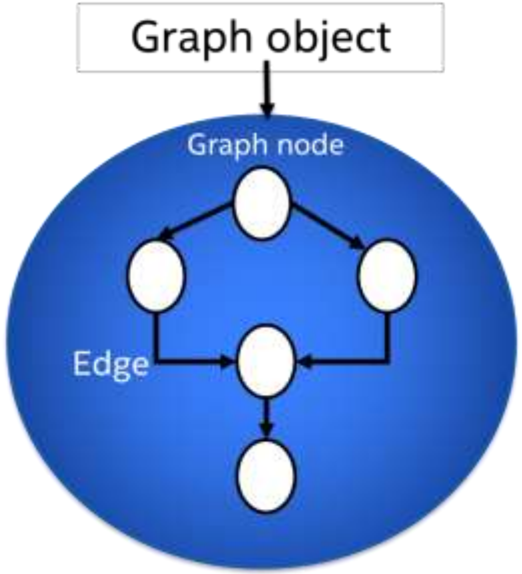
TASK PARALLLISM: INTEL TBB FLOW GRAPH

- Introduced in Intel TBB 4.0
- Efficient implementation of dependency graph and data flow algorithms
- Initially Designed for shared memory application
- Enables developers to exploit parallelism at higher levels

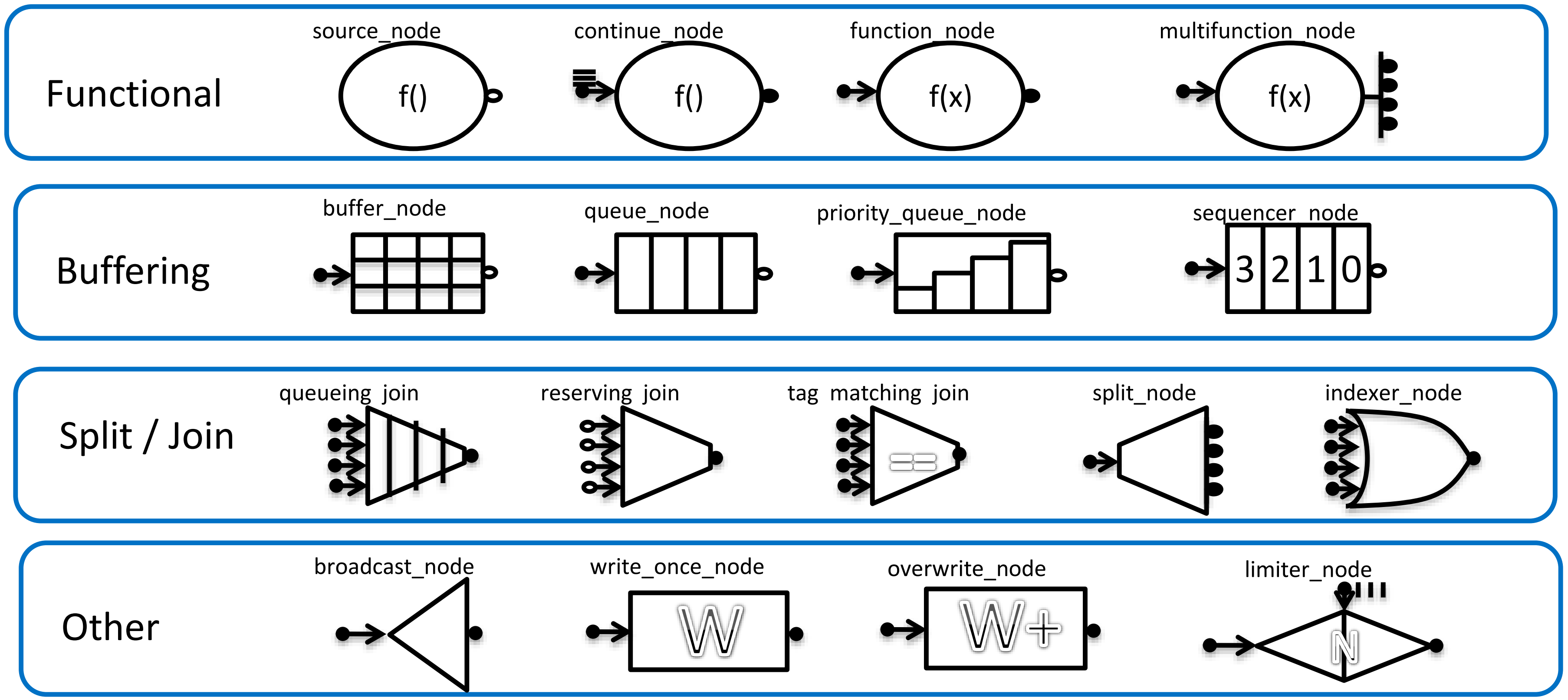
Hello World



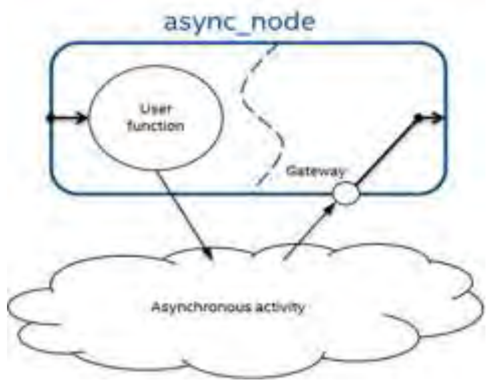
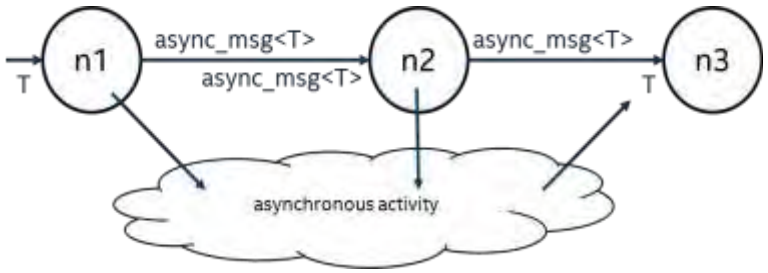
```
graph g;
continue_node< continue_msg > h( g,
    []( const continue_msg & ){              cout <<
        “Hello “;
    });
continue_node< continue_msg > w( g,
    []( const continue_msg & ){
        cout << “World\n“;
    } );
make_edge( h, w );
h.try_put(continue_msg());
g.wait_for_all();
```



INTEL TBB FLOW GRAPH NODE TYPES



INTEL TBB FLOW GRAPH FOR HETEROGENEOUS PROGRAMING

Feature	Description	Diagram
async_node<Input,Output> *Available as production feature*	Basic building block. Enables async communication from a single/isolated node to an async activity. User responsible for managing communication. Graph runs on host.	 async_node User function Gateway Asynchronous activity
async_msg<T> *Available as preview feature*	Basic building block. Enables async communication with chaining across graph nodes. User responsible for managing communication. Graph runs on the host.	 T n1 async_msg<T> async_msg<T> n2 async_msg<T> T n3 asynchronous activity

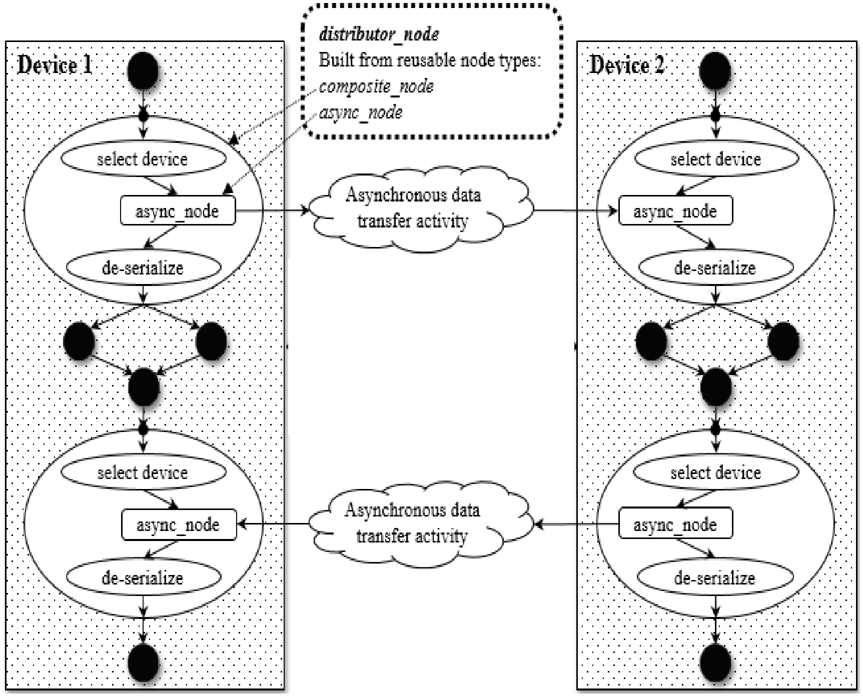
THE DISTRIBUTOR_NODE (EXPERIMENTAL)

Enables communication between different memory domains. Each device is capable of running a graph; e.g. hosts, co-processor, etc…

Graphs is execute on all devices

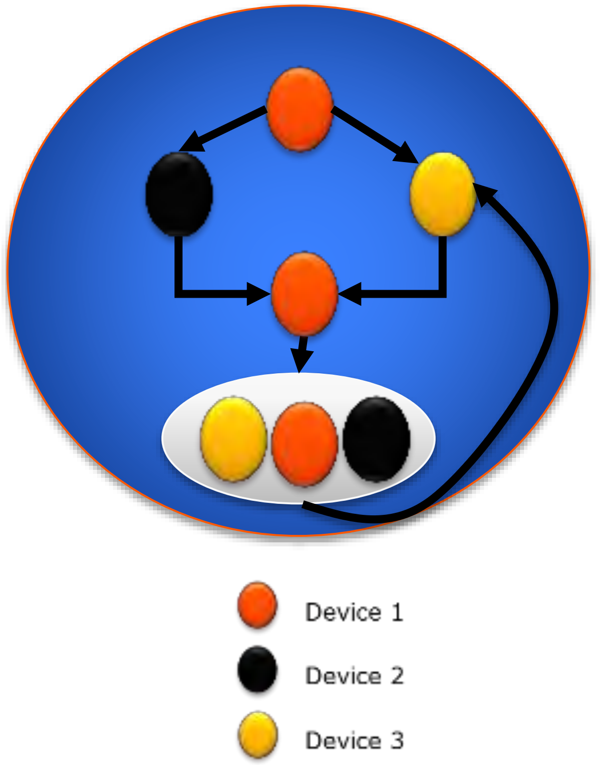
Communication can be initiated from any device to any device

Sub-graphs are executed on a device between communication points



HETEROGINITY: INTEL FLOW GRAPH AS COORDINATION LAYER

- Exposes parallelism between blocks; simplifies integration
- The glue that connects distributed HW and SW IP
- Libraries implemented using Intel® TBB will compose seamlessly
- Dynamic load balancing
- Supports Distributed memory



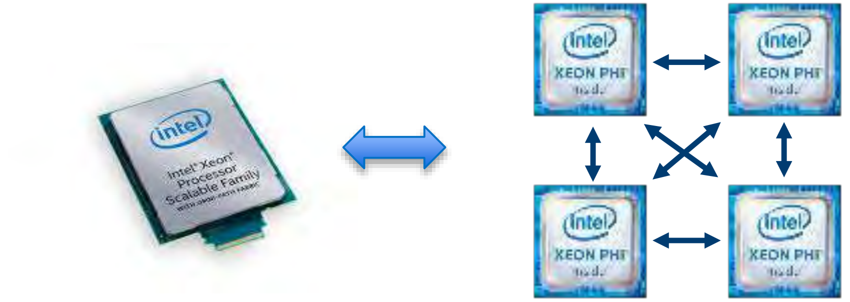
STAC-A2* - THE FINANCIAL INDUSTRY BENCHMARK

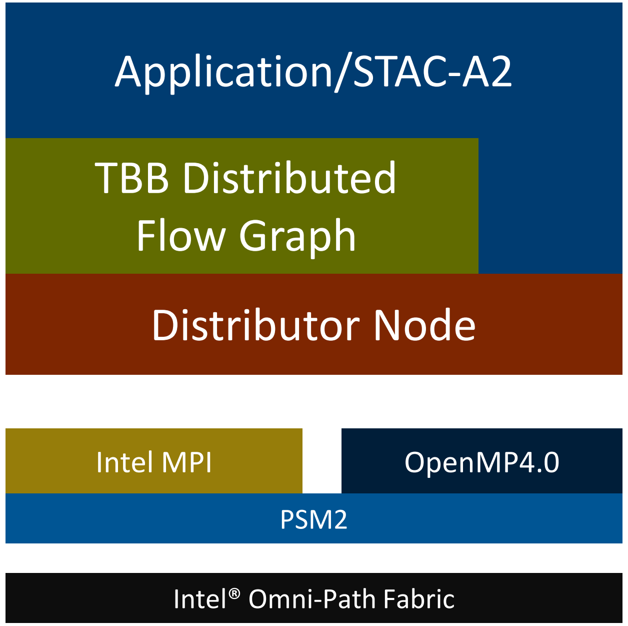
- The STAC-A2 Benchmark is a suite for testing technology platforms (SUT)
- STAC-A2 specifications are based on Market Risk analysis
- "Customers" define the specifications, "Vendors" implement the code
- 7 types of Greeks
- Complex Multi-Asset pricing of American Options
  - Heston Stochastic Volatility underlying price model
  - Least-Squares Monte Carlo / Longstaff-Schwartz algorithm
- Intel first published the benchmark results in Supercomputing'12

* "STAC" and all STAC names are trademarks or registered trademarks of the Securities Technology Analysis Center LLC.

STAC-A2* INTEL IMPLEMENTATION HIGHLIGHTS

- Intel® Parallel Studio components used:
  - Intel Compiler & OpenMP 4.0 for vectorization
  - Intel TBB parallel algorithms for for-join constructs
  - Intel TBB Flow Graph for task distribution
  - Intel® Math Kernel Library (Intel® MKL)
- distributor_node for distributed compute
- A token-based approach for dynamic load balancing between the main CPU and "coprocessors"





BREAKING DEPENDENCIES IN MONTE-CARLO (STOCHASTIC VOL)

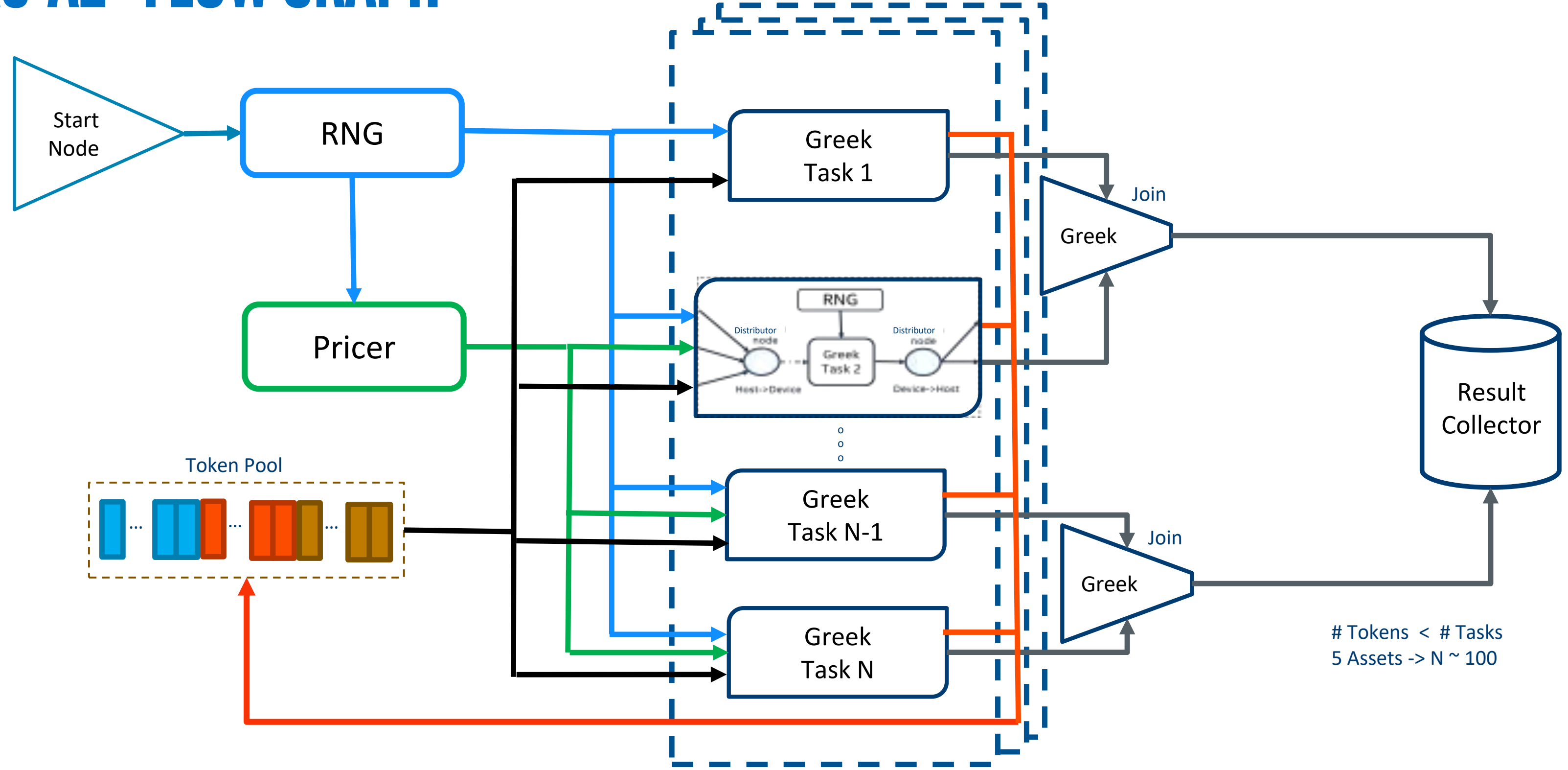
```
for (unsigned p = 0; i < nPaths; ++p)
{
   double mV[nTimeSteps]; // Spot Volatility state
   double mY[nTimeSteps]; // Spot Price state
    .....
    for (unsigned int t = 0; t < nTimeSteps; ++t){
      double logSpotPrice = func(mY[t], mV[t]); // Backward dependency
      mY[t+1] = logSpotPrice * A[t];
      mV[t+1] = logSpotPrice * B[t] + C[t] * mV[t];
      price[p][t] = exp(mY[t]);
  }
}
```

BREAKING DEPENDENCIES IN MONTE-CARLO (STOCHASTIC VOL)

```
for (unsigned p = 0; i < nPaths; ++p)
{
   double mV[nTimeSteps]; // Spot Volatility state
   double mY[nTimeSteps]; // Spot Price state
     .....
     for (unsigned int t = 0; t < nTimeSteps; ++t){
     double logSpotPrice = func(mY[t], mV[t]); // Backward
dependency
     mY[t+1] = logSpotPrice * A[t];
     mV[t+1] = logSpotPrice * B[t] + C[t] * mV[t];
     price[p][t] = exp(mY[t]);
  }
}
```

```
tbb:parallel_for(tbb::blocked_range<int>(0, nPaths, 256),
   [&](const tbb::blocked_range<int>& r) {
    const auto block_size = r.size();
    double mV[nTimeSteps][block_size]; // Spot Volatility sta
    double mY[nTimeSteps][block_size]; // Spot Price state
     for (unsigned int t = 0; t < nTimeSteps; ++t){
       for (unsigned p = 0; i < nPaths; ++p)
       {
         double logSpotPrice = func(mY[t][p], mV[t][p]);
         mY[t+1][p] = logSpotPrice * A[t];
         mV[t+1][p] = logSpotPrice * B[t] + C[t] * mV[t][p];
         price[r.begin()+p][t] = exp(mY[t][p]);
   }
}
```

STAC-A2* FLOW GRAPH



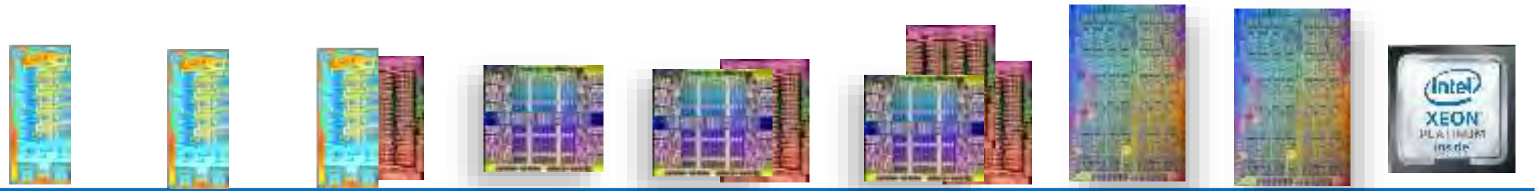
# Tokens < # Tasks
5 Assets -> N ~ 100

HETEROGENEOUS GREEK NODE

```
#pragma offload_attribute(push, target(mic))
typedef execution_node  < tbb::flow::tuple<std::shared_ptr<GreekResults>, device_token_t >, double> execution_node_theta_t;
…
void CreateGraph(…) {
…
theta_node = std::make_shared<execution_node_theta_t>(_g,
[arena, pWS, randoms](const std::shared_ptr<GreekResults>&, const device_token_t& t) -> double {
        double pv = 0.;
        std::shared_ptr<ArrayContainer<double>> unCorrRandomNumbers;
        randoms->try_get(unCorrRandomNumbers);
        const double deltaT = 1.0 / 100.0;
        pv = f_scenario_adj<false>(pWS->r, …, pWS->A, unCorrRandomNumbers);
        return pv;
    }
, true));
…
}
#pragma offload_attribute(pop)
```

Same code executed on Xeon and Xeon Phi, Enabled by Intel Compiler

EVOLUTION STAC-A2 – 37X OVER 4 YEARS



	Intel Xeon processor E5 2697-V2	Intel Xeon processor E5 2697-V2	Intel Xeon E5 2697-V2 + Xeon Phi	Intel Xeon E5 2697-V3	Intel Xeon E5 2697-V3+ Xeon Phi	Intel Xeon E5 2697-V3+ 2*Xeon Phi	Intel Xeon Phi 7290	Intel Xeon Phi 7290	Intel Xeon Platinum 8180
	2013	2014	2014	2014	2014	2015	2016	2017	2017
cores	24	24	24+61	36	36+61	36+122	72	72	56
Threads	48	48	48+244	72	72+244	72+488	288	288	112
Vectors(Size)	256	256	256+512	256	256+512	256+2*512	512	512	512
Parallelization	OpenMP →	TBB	TBB	TBB	TBB	TBB	TBB	TBB	TBB
Vectorization	#SIMD	OpenMP	OpenMP	OpenMP	OpenMP	OpenMP	OpenMP	OpenMP	OpenMP
Heterogeneity	N/A	N/A →	OpenMP	N/A	OpenMP →	TBB	N/A	N/A	N/A
Greek time	4.8 →	1.0 →	0.63	0.81	0.53 →	0.216	0.207 →	0.141	0.128

Appropriate Threading model

1st Heterogeneous Implementation

Dynamic Load Balancing between 3 devices

Improved Memory accesses(SDLT)

SUMMARY

- Identify 3 levels of parallelism to leverage ALL available HW resources
- Intel TBB allows compasability and utilization of the compute resources
- Intel TBB Flow Graph extensions allow task distribution & dynamic load balancing
- Intel tools support C/C++ for programing in heterogeneous environment

CODE THAT PERFORMS AND OUTPERFORMS

Download a *free*, 30-day trial of
Intel® Parallel Studio XE 2018 today

https://software.intel.com/en-us/intel-parallel-studio-xe/try-buy

AND DON'T FORGET...

To check your inbox for the evaluation survey which will be emailed after this presentation.

P.S.

Everyone who fills out the survey will receive a personalized certificate indicating completion of the training!

LEGAL DISCLAIMERS

intel®

Software