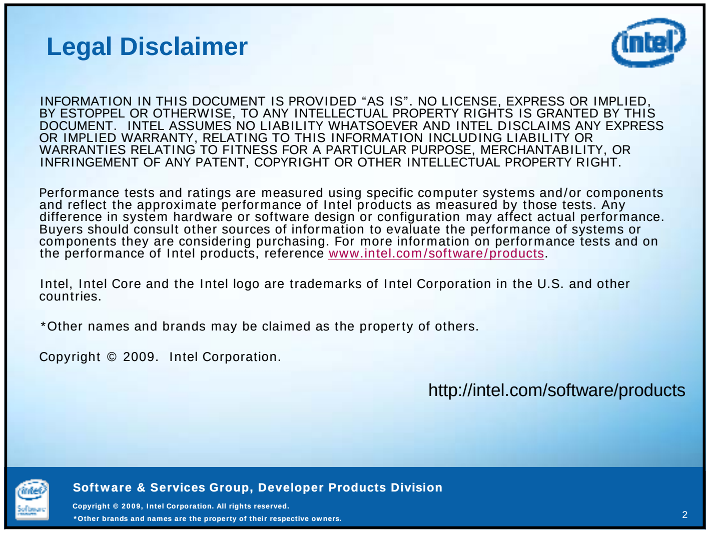
Ct: *A New Paradigm for Data Parallel Computing*

Hans–Christian Hoppe
Intel Visual Computing Institute, Intel Labs

using material from

Anwar Ghuloum, CJ Newburn, Michael McCool and Stefanus Du Toit
Performance and Productivity Libraries, Developer Products Division,
Software and Services Group

PRACE Partnership for Advanced Computing in Europe



Legal Disclaimer

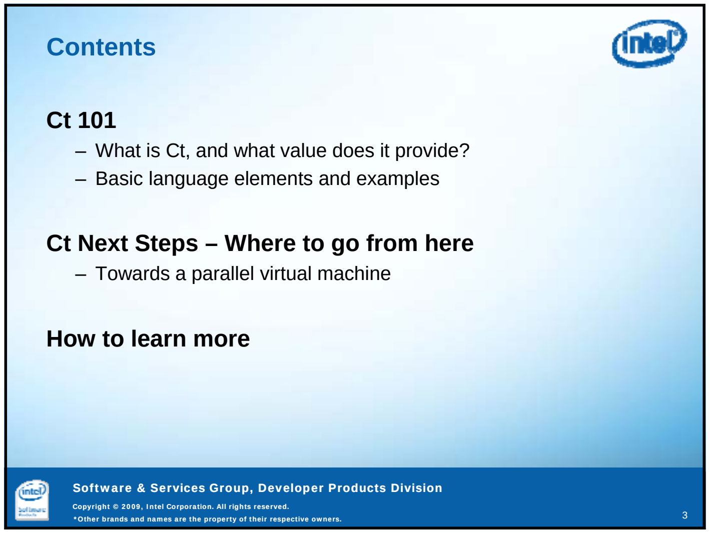
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

Intel, Intel Core and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

http://intel.com/software/products



Contents

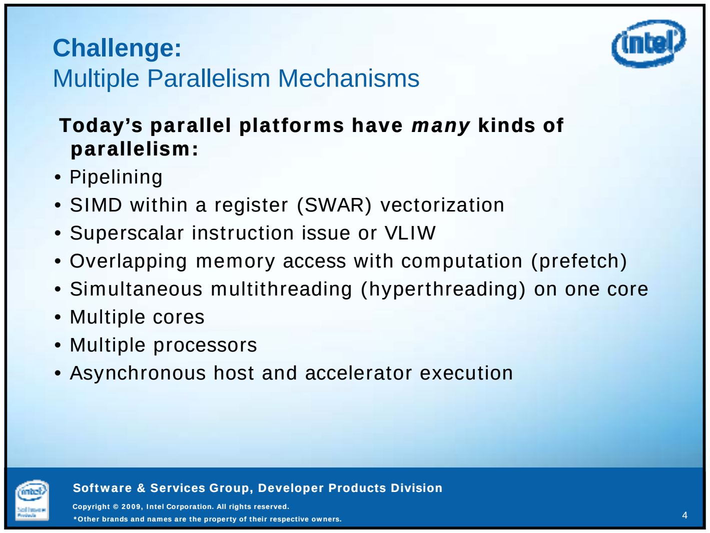
Ct 101

- What is Ct, and what value does it provide?
- Basic language elements and examples

Ct Next Steps – Where to go from here

- Towards a parallel virtual machine

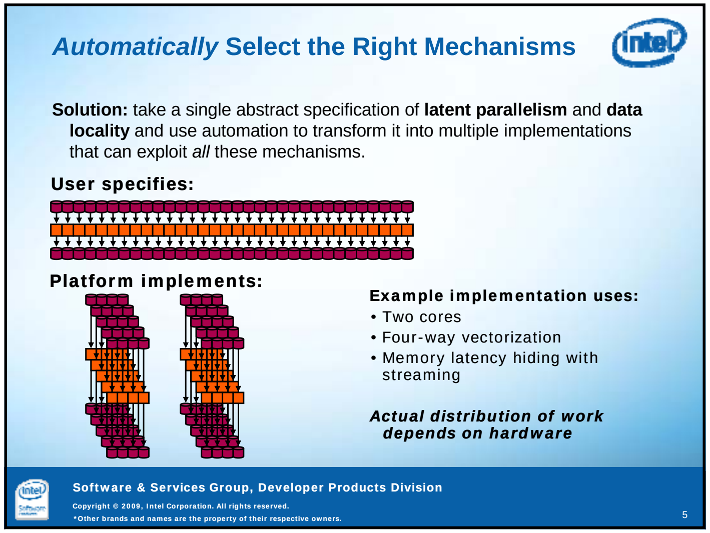
How to learn more



Challenge:
Multiple Parallelism Mechanisms

Today's parallel platforms have *many* kinds of parallelism:

- Pipelining
- SIMD within a register (SWAR) vectorization
- Superscalar instruction issue or VLIW
- Overlapping memory access with computation (prefetch)
- Simultaneous multithreading (hyperthreading) on one core
- Multiple cores
- Multiple processors
- Asynchronous host and accelerator execution



Automatically Select the Right Mechanisms

Solution: take a single abstract specification of **latent parallelism** and **data locality** and use automation to transform it into multiple implementations that can exploit *all* these mechanisms.

User specifies:

Platform implements:

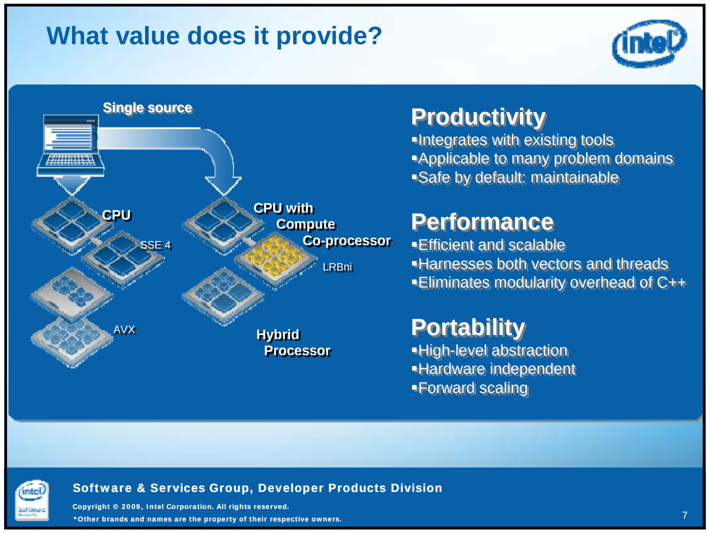
Example implementation uses:

- Two cores
- Four-way vectorization
- Memory latency hiding with streaming

Actual distribution of work depends on hardware

What is Ct Technology?

- A generalized data parallel programming solution that frees application developers from dependencies on particular hardware architectures.
- A system that integrates with existing development tools to allow parallel algorithms to be specified at a high level.
- A dynamic compiler and runtime that translates high-level specifications of computations into efficient parallel implementations that can take advantage of both SIMD and thread-level parallelism, as well as accelerators.
- ***A system that allows an application developer to combine performance, portability, and productivity.***

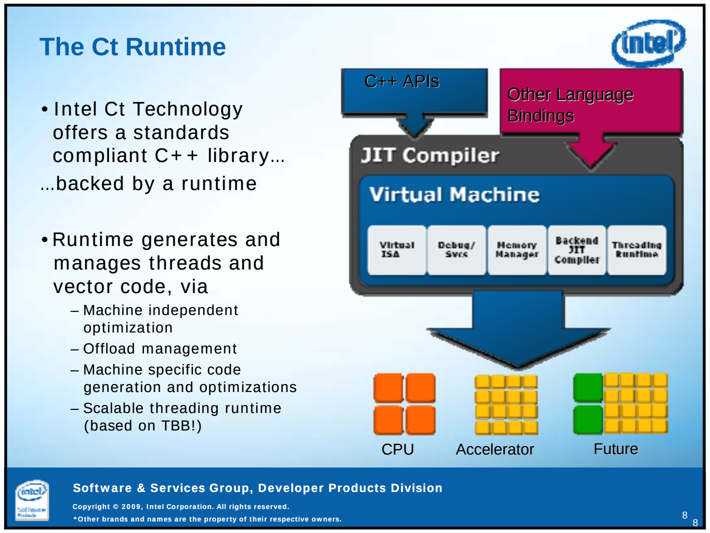


What value does it provide?

Productivity
- Integrates with existing tools
- Applicable to many problem domains
- Safe by default: maintainable

Performance
- Efficient and scalable
- Harnesses both vectors and threads
- Eliminates modularity overhead of C++

Portability
- High-level abstraction
- Hardware independent
- Forward scaling

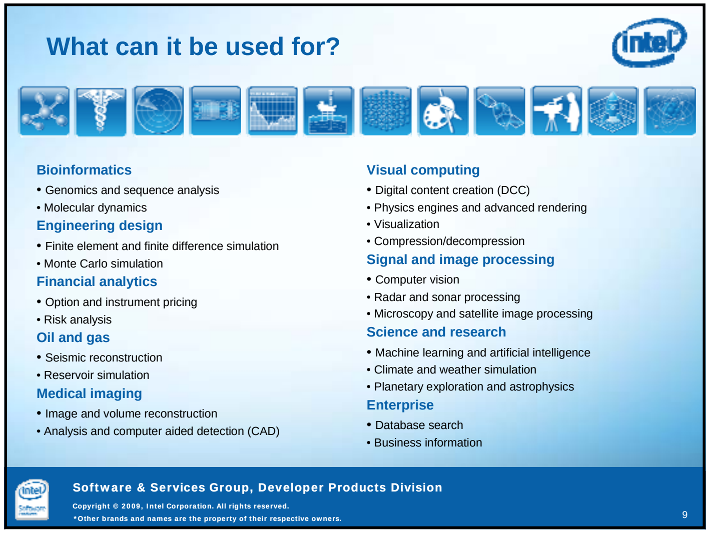


The Ct Runtime

- Intel Ct Technology offers a standards compliant C++ library…

…backed by a runtime

- Runtime generates and manages threads and vector code, via
  - Machine independent optimization
  - Offload management
  - Machine specific code generation and optimizations
  - Scalable threading runtime (based on TBB!)



What can it be used for?

Bioinformatics
- Genomics and sequence analysis
- Molecular dynamics

Engineering design
- Finite element and finite difference simulation
- Monte Carlo simulation

Financial analytics
- Option and instrument pricing
- Risk analysis

Oil and gas
- Seismic reconstruction
- Reservoir simulation

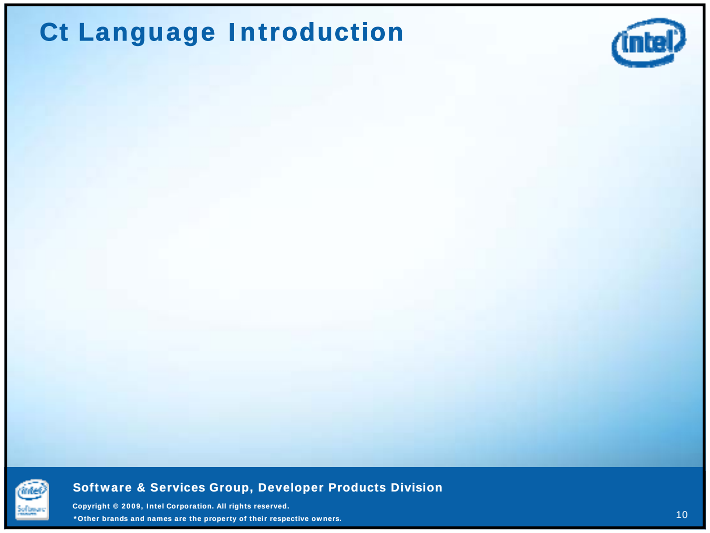
Medical imaging
- Image and volume reconstruction
- Analysis and computer aided detection (CAD)

Visual computing
- Digital content creation (DCC)
- Physics engines and advanced rendering
- Visualization
- Compression/decompression

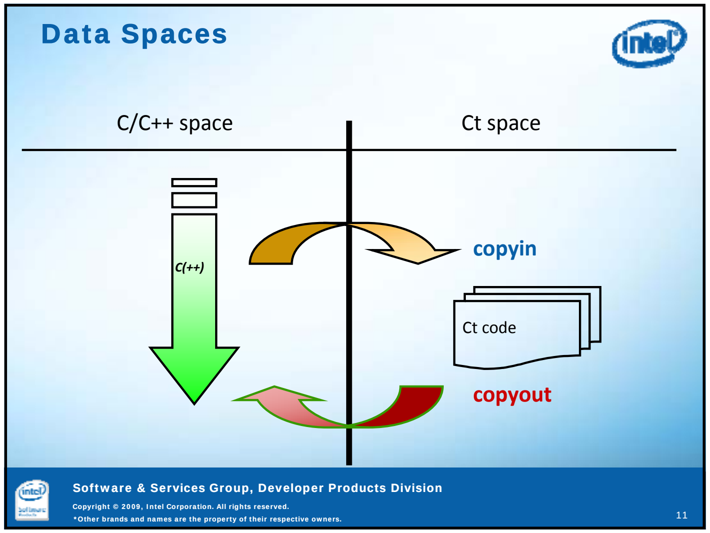
Signal and image processing
- Computer vision
- Radar and sonar processing
- Microscopy and satellite image processing

Science and research
- Machine learning and artificial intelligence
- Climate and weather simulation
- Planetary exploration and astrophysics

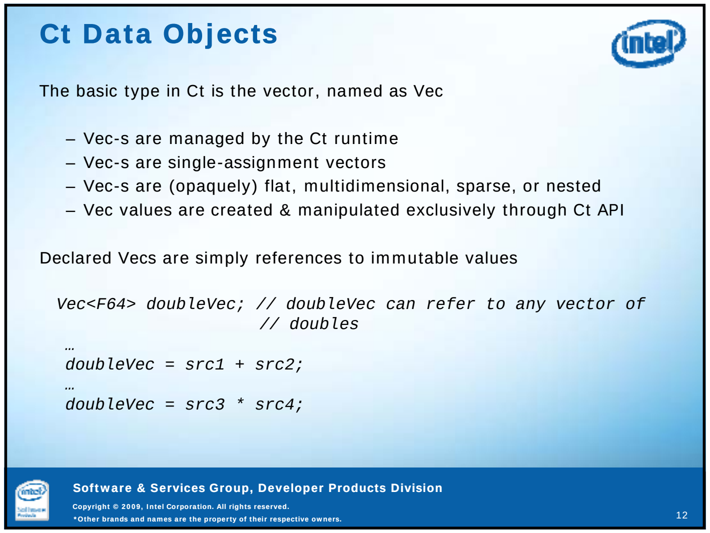
Enterprise
- Database search
- Business information



Ct Language Introduction



Data Spaces



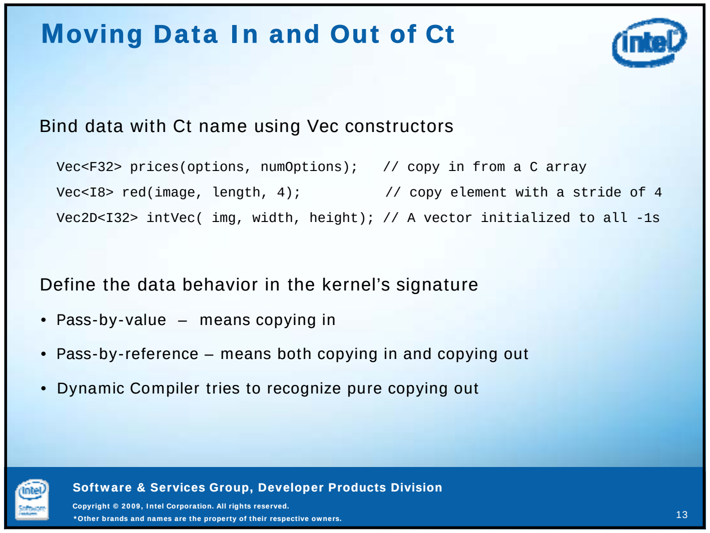
Ct Data Objects

The basic type in Ct is the vector, named as Vec

- Vec-s are managed by the Ct runtime
- Vec-s are single-assignment vectors
- Vec-s are (opaquely) flat, multidimensional, sparse, or nested
- Vec values are created & manipulated exclusively through Ct API

Declared Vecs are simply references to immutable values

```
Vec<F64> doubleVec; // doubleVec can refer to any vector of
                    // doubles
 …
 doubleVec = src1 + src2;
 …
 doubleVec = src3 * src4;
```



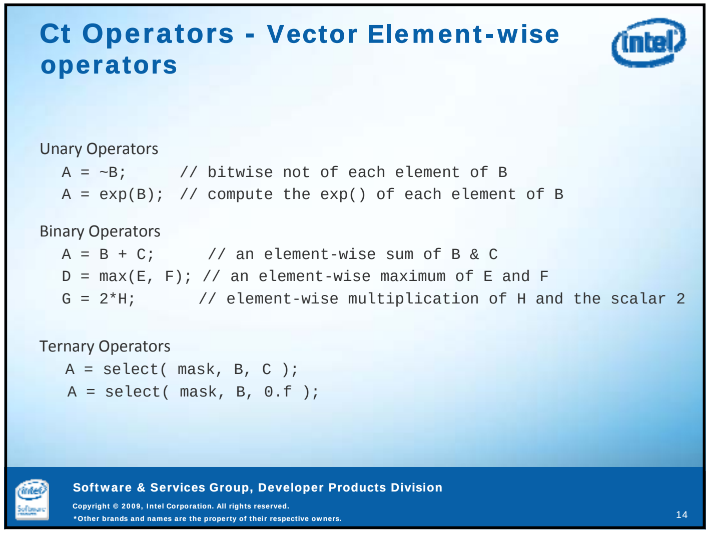
Moving Data In and Out of Ct

Bind data with Ct name using Vec constructors

```
Vec<F32> prices(options, numOptions);   // copy in from a C array
Vec<I8> red(image, length, 4);          // copy element with a stride of 4
Vec2D<I32> intVec( img, width, height); // A vector initialized to all -1s
```

Define the data behavior in the kernel's signature

- Pass-by-value – means copying in
- Pass-by-reference – means both copying in and copying out
- Dynamic Compiler tries to recognize pure copying out



Ct Operators - Vector Element-wise operators

Unary Operators

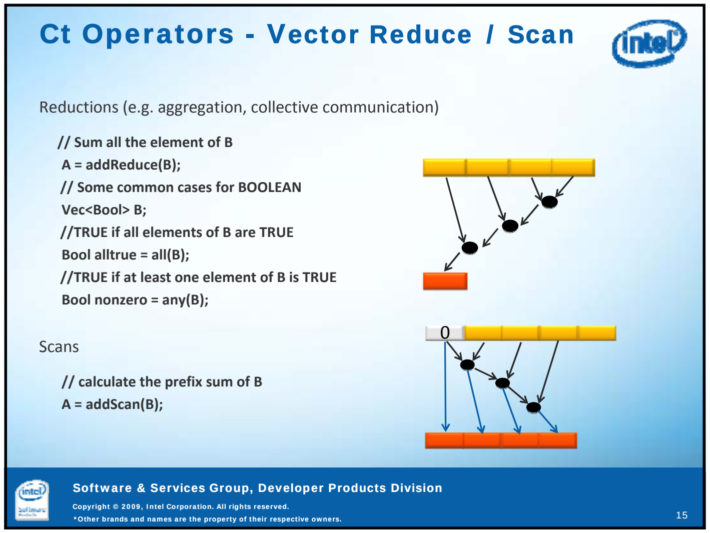
```
A = ~B;      // bitwise not of each element of B
A = exp(B);  // compute the exp() of each element of B
```

Binary Operators

```
A = B + C;      // an element-wise sum of B & C
D = max(E, F); // an element-wise maximum of E and F
G = 2*H;       // element-wise multiplication of H and the scalar 2
```

Ternary Operators

```
A = select( mask, B, C );
A = select( mask, B, 0.f );
```



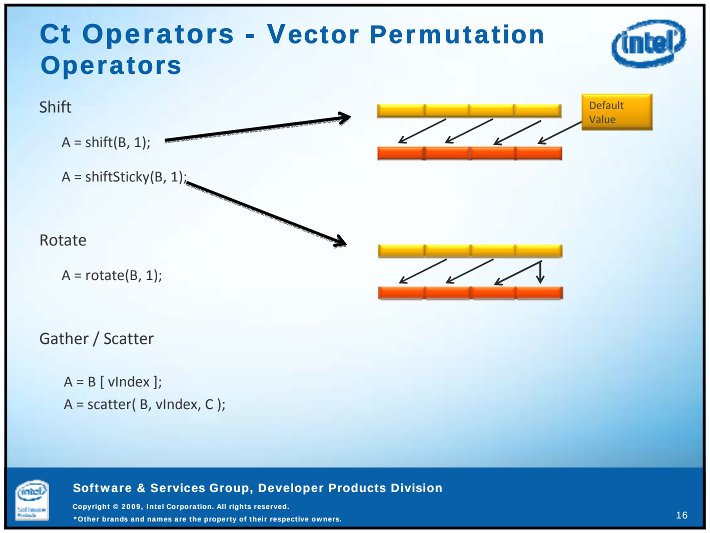
Ct Operators - Vector Reduce / Scan

Reductions (e.g. aggregation, collective communication)

```
// Sum all the element of B
A = addReduce(B);
// Some common cases for BOOLEAN
Vec<Bool> B;
//TRUE if all elements of B are TRUE
Bool alltrue = all(B);
//TRUE if at least one element of B is TRUE
Bool nonzero = any(B);
```

Scans

```
// calculate the prefix sum of B
A = addScan(B);
```



Ct Operators - Vector Permutation Operators

Shift

A = shift(B, 1);

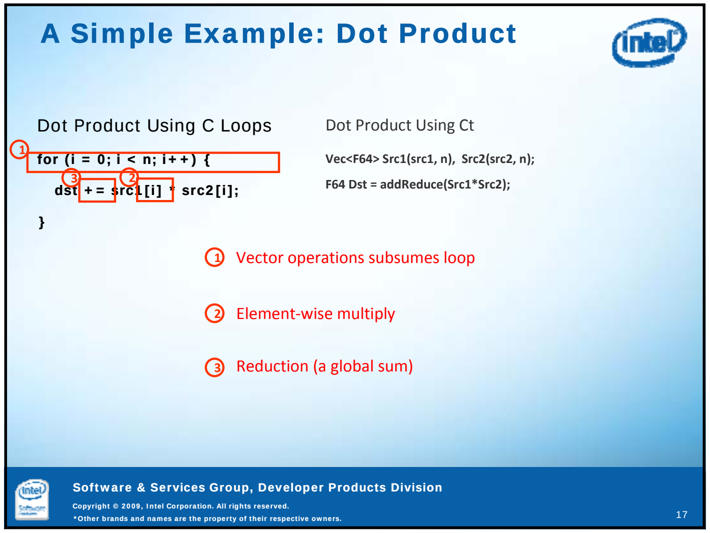
A = shiftSticky(B, 1);

Rotate

A = rotate(B, 1);

Gather / Scatter

A = B [vIndex];
A = scatter(B, vIndex, C);



A Simple Example: Dot Product

Dot Product Using C Loops

```
① for (i = 0; i < n; i++) {
     dst ③+= src1②[i] * src2[i];
  }
```

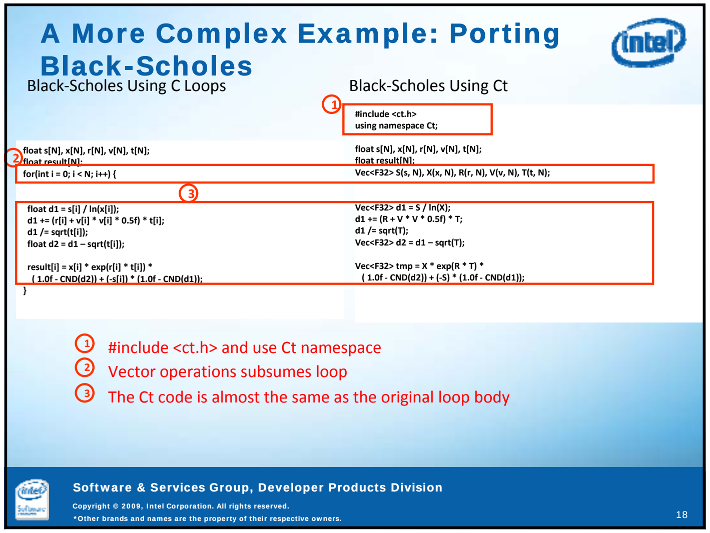
Dot Product Using Ct

```
Vec<F64> Src1(src1, n),  Src2(src2, n);

F64 Dst = addReduce(Src1*Src2);
```

① Vector operations subsumes loop

② Element-wise multiply

③ Reduction (a global sum)



A More Complex Example: Porting Black-Scholes

Black-Scholes Using C Loops

```
float s[N], x[N], r[N], v[N], t[N];
float result[N];
for(int i = 0; i < N; i++) {

  float d1 = s[i] / ln(x[i]);
  d1 += (r[i] + v[i] * v[i] * 0.5f) * t[i];
  d1 /= sqrt(t[i]);
  float d2 = d1 – sqrt(t[i]);

  result[i] = x[i] * exp(r[i] * t[i]) *
    ( 1.0f - CND(d2)) + (-s[i]) * (1.0f - CND(d1));
}
```

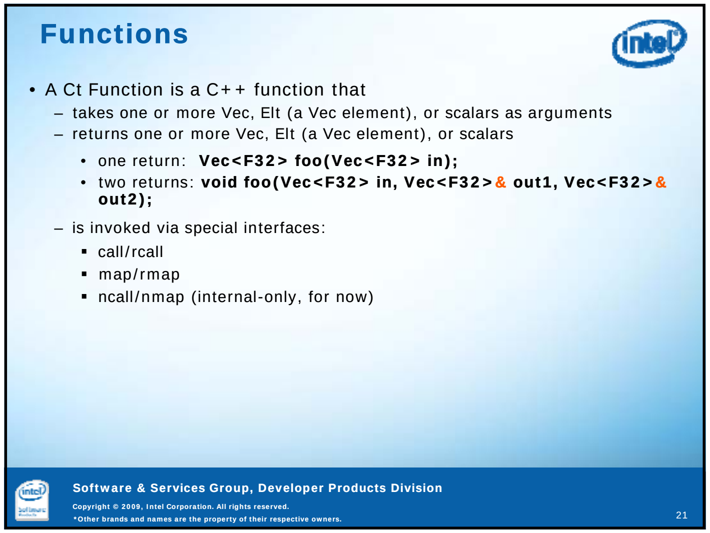
Black-Scholes Using Ct

```
#include <ct.h>
using namespace Ct;

float s[N], x[N], r[N], v[N], t[N];
float result[N];
Vec<F32> S(s, N), X(x, N), R(r, N), V(v, N), T(t, N);

Vec<F32> d1 = S / ln(X);
d1 += (R + V * V * 0.5f) * T;
d1 /= sqrt(T);
Vec<F32> d2 = d1 – sqrt(T);

Vec<F32> tmp = X * exp(R * T) *
  ( 1.0f - CND(d2)) + (-S) * (1.0f - CND(d1));
```

① #include <ct.h> and use Ct namespace

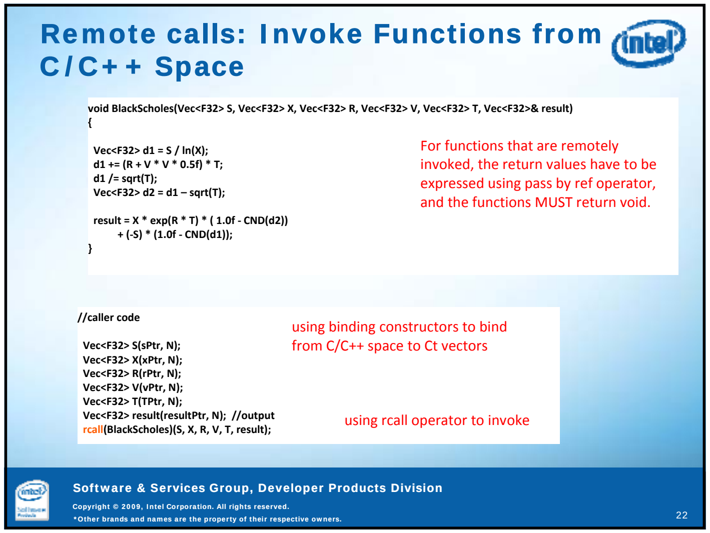
② Vector operations subsumes loop

③ The Ct code is almost the same as the original loop body



Functions

- A Ct Function is a C++ function that
  - takes one or more Vec, Elt (a Vec element), or scalars as arguments
  - returns one or more Vec, Elt (a Vec element), or scalars
    - one return: **Vec<F32> foo(Vec<F32> in);**
    - two returns: **void foo(Vec<F32> in, Vec<F32>& out1, Vec<F32>& out2);**
  - is invoked via special interfaces:
    - call/rcall
    - map/rmap
    - ncall/nmap (internal-only, for now)



Remote calls: Invoke Functions from C/C++ Space

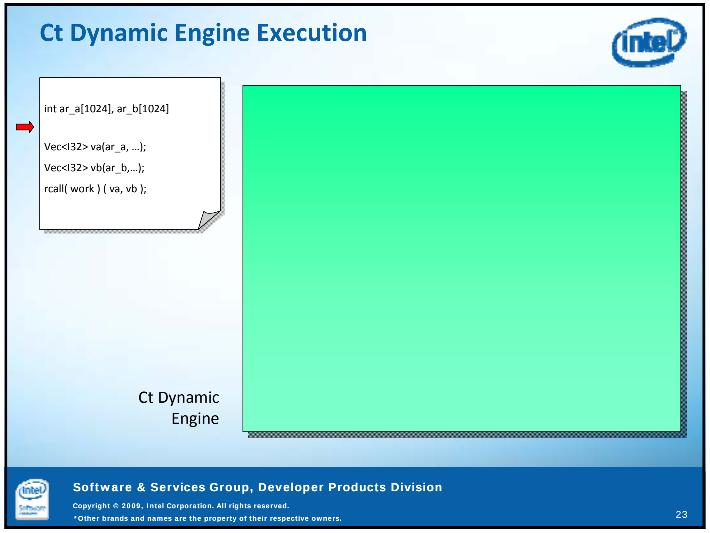
```
void BlackScholes(Vec<F32> S, Vec<F32> X, Vec<F32> R, Vec<F32> V, Vec<F32> T, Vec<F32>& result)
{

 Vec<F32> d1 = S / ln(X);
 d1 += (R + V * V * 0.5f) * T;
 d1 /= sqrt(T);
 Vec<F32> d2 = d1 – sqrt(T);

 result = X * exp(R * T) * ( 1.0f - CND(d2))
        + (-S) * (1.0f - CND(d1));
}
```

For functions that are remotely invoked, the return values have to be expressed using pass by ref operator, and the functions MUST return void.

```
//caller code

 Vec<F32> S(sPtr, N);
 Vec<F32> X(xPtr, N);
 Vec<F32> R(rPtr, N);
 Vec<F32> V(vPtr, N);
 Vec<F32> T(TPtr, N);
 Vec<F32> result(resultPtr, N);  //output
 rcall(BlackScholes)(S, X, R, V, T, result);
```

using binding constructors to bind from C/C++ space to Ct vectors

using rcall operator to invoke

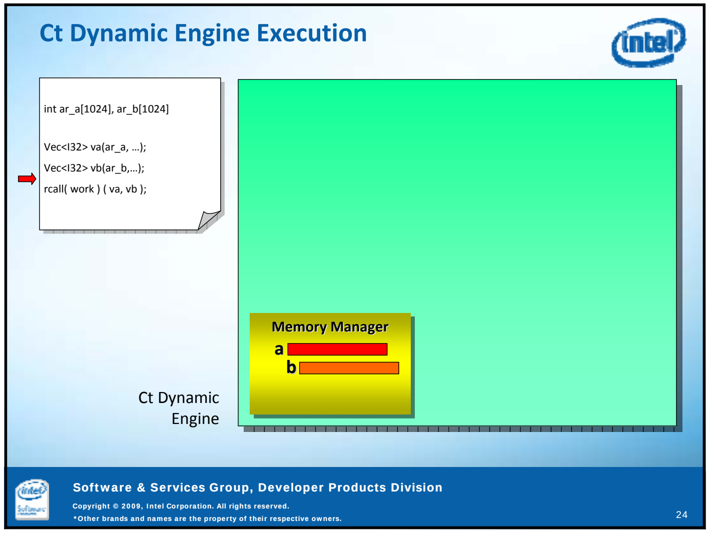


Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

Ct Dynamic Engine

Software & Services Group, Developer Products Division



Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

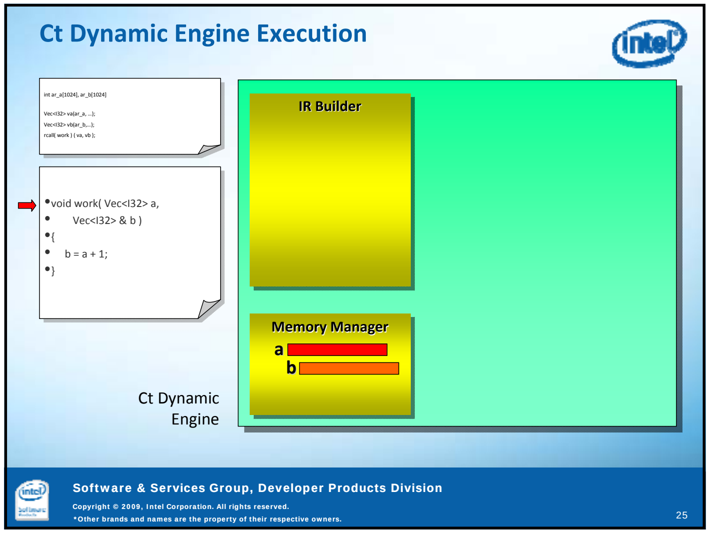
Memory Manager

a

b

Ct Dynamic Engine

Software & Services Group, Developer Products Division



Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

```
•void work( Vec<I32> a,
•      Vec<I32> & b )
•{
•    b = a + 1;
•}
```

IR Builder

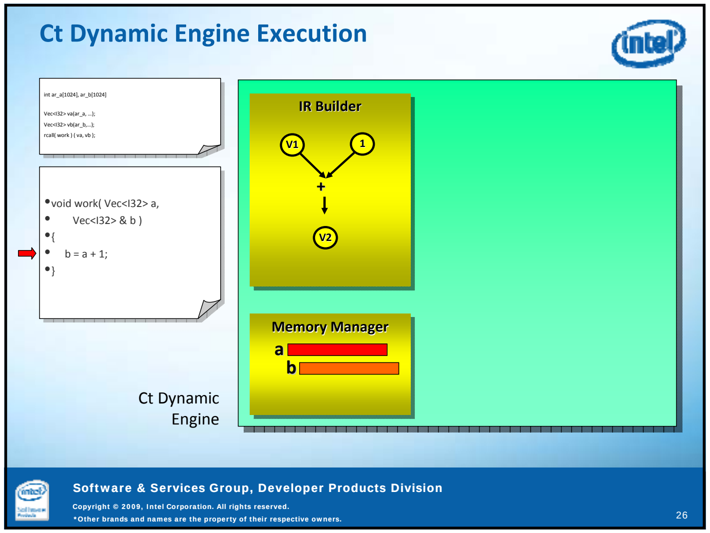
Memory Manager

a

b

Ct Dynamic Engine

Software & Services Group, Developer Products Division



Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

```
•void work( Vec<I32> a,
•      Vec<I32> & b )
•{
•    b = a + 1;
•}
```

IR Builder

Memory Manager

a

b

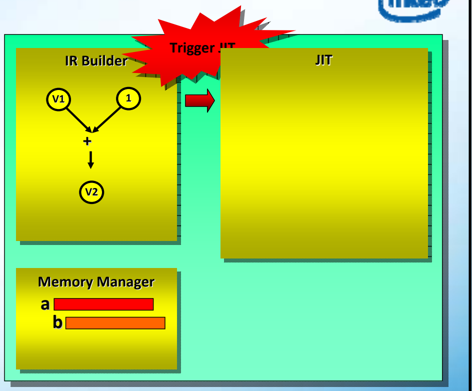
Ct Dynamic Engine

Software & Services Group, Developer Products Division

Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

```
• void work( Vec<I32> a,
•       Vec<I32> & b )
• {
•    b = a + 1;
• }
```

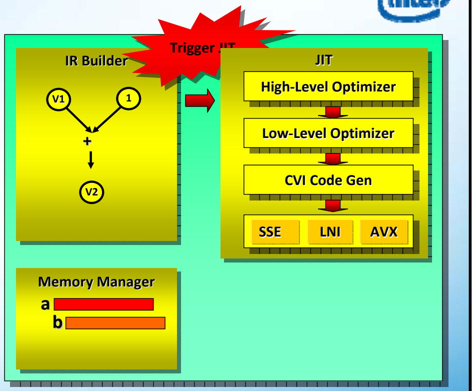


Ct Dynamic
Engine

Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

```
• void work( Vec<I32> a,
•       Vec<I32> & b )
• {
•    b = a + 1;
• }
```

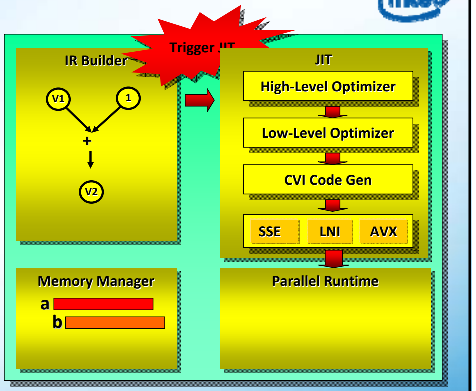


Ct Dynamic
Engine

Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

```
• void work( Vec<I32> a,
•       Vec<I32> & b )
• {
•    b = a + 1;
• }
```

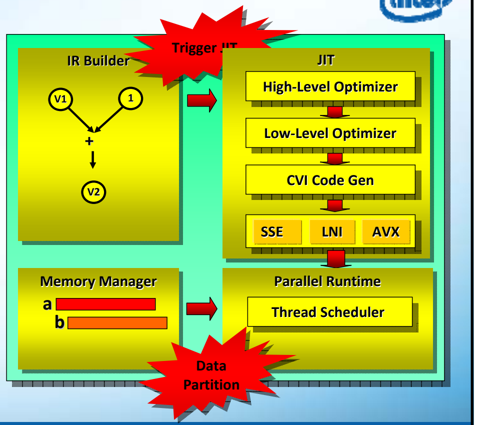


Ct Dynamic
Engine

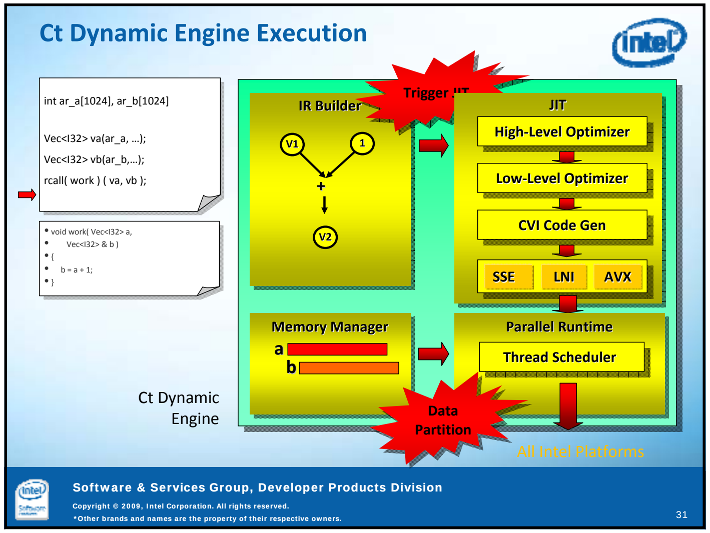
Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

```
• void work( Vec<I32> a,
•       Vec<I32> & b )
• {
•    b = a + 1;
• }
```



Ct Dynamic
Engine



Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

```
• void work( Vec<I32> a,
•       Vec<I32> & b )
• {
•    b = a + 1;
• }
```

IR Builder

V1

1

+

V2

Trigger JIT

JIT

High-Level Optimizer

Low-Level Optimizer

CVI Code Gen

SSE

LNI

AVX

Memory Manager

a

b

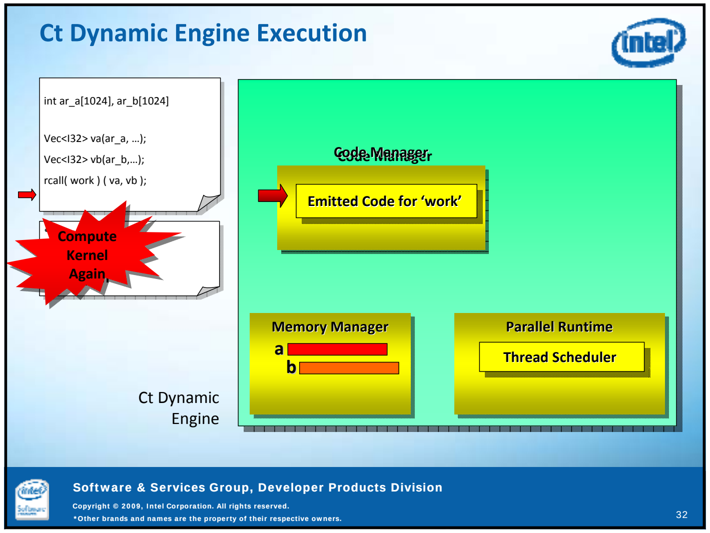
Parallel Runtime

Thread Scheduler

Data Partition

Ct Dynamic Engine

All Intel Platforms



Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

Compute Kernel Again

Code Manager

Emitted Code for ‘work’

Memory Manager

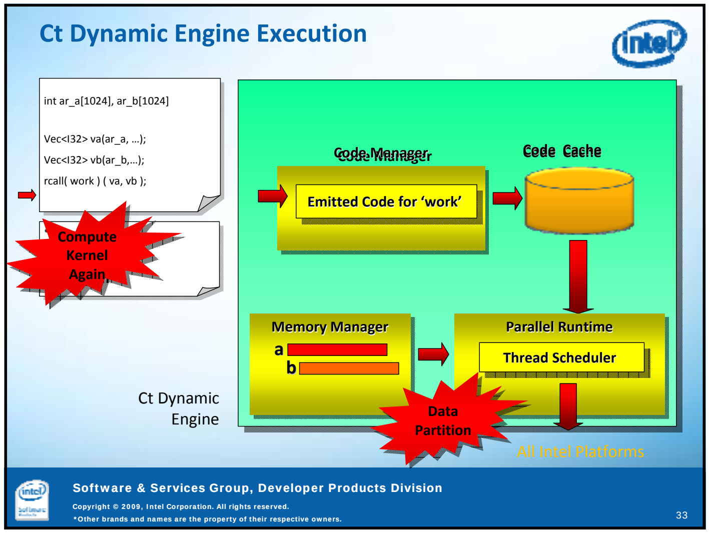
a

b

Parallel Runtime

Thread Scheduler

Ct Dynamic Engine



Ct Dynamic Engine Execution

```
int ar_a[1024], ar_b[1024]

Vec<I32> va(ar_a, …);
Vec<I32> vb(ar_b,…);
rcall( work ) ( va, vb );
```

Compute Kernel Again

Code Manager

Emitted Code for ‘work’

Code Cache

Memory Manager

a

b

Parallel Runtime

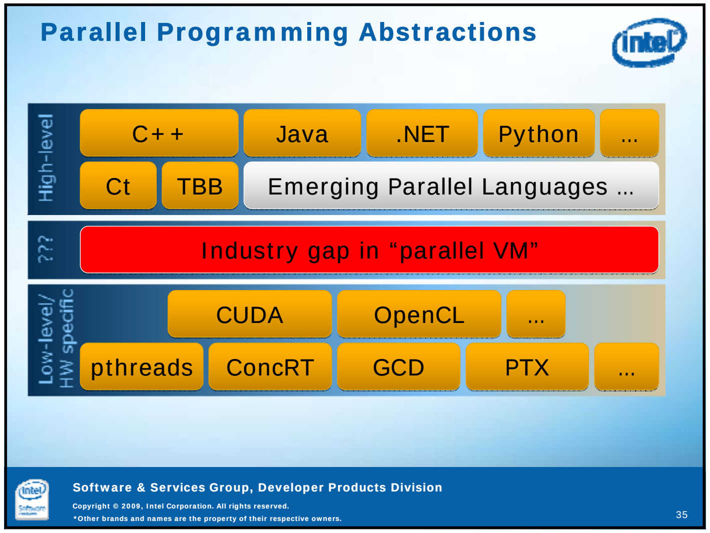
Thread Scheduler

Data Partition

Ct Dynamic Engine

All Intel Platforms

Towards a Parallel Virtual Machine



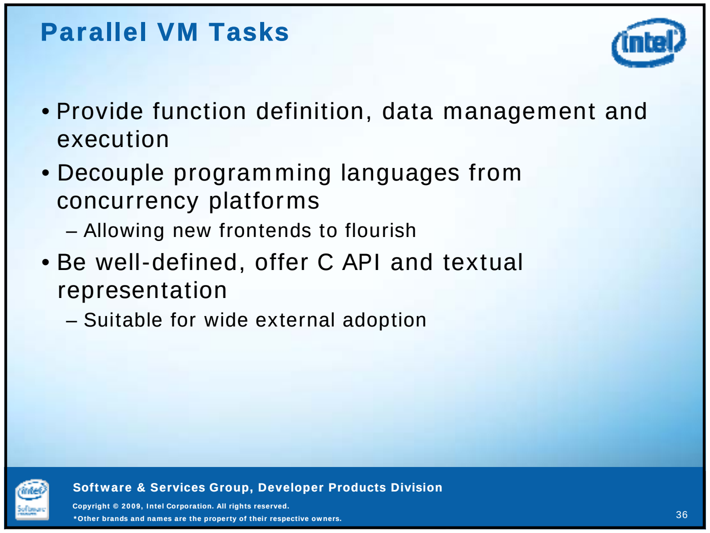
Parallel Programming Abstractions

High-level
- C++
- Java
- .NET
- Python
- ...
- Ct
- TBB
- Emerging Parallel Languages ...

???
- Industry gap in “parallel VM”

Low-level/ HW specific
- CUDA
- OpenCL
- ...
- pthreads
- ConcRT
- GCD
- PTX
- ...

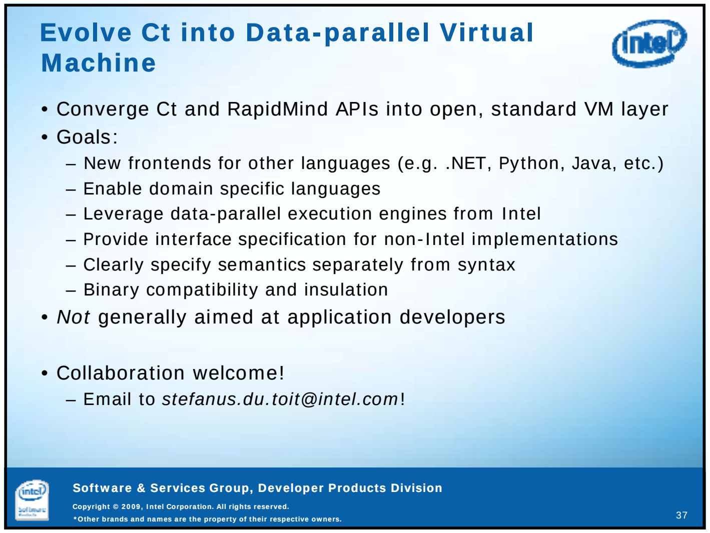
Software & Services Group, Developer Products Division



Parallel VM Tasks

- Provide function definition, data management and execution
- Decouple programming languages from concurrency platforms
  - Allowing new frontends to flourish
- Be well-defined, offer C API and textual representation
  - Suitable for wide external adoption

Software & Services Group, Developer Products Division

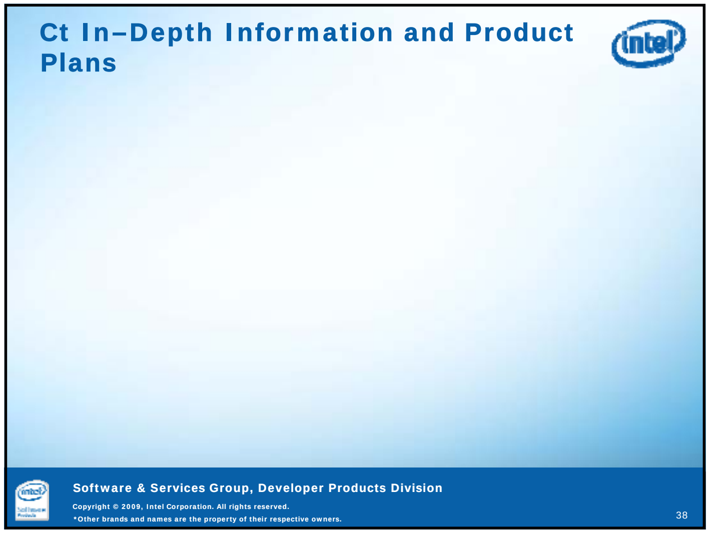


Evolve Ct into Data-parallel Virtual Machine

- Converge Ct and RapidMind APIs into open, standard VM layer
- Goals:
  - New frontends for other languages (e.g. .NET, Python, Java, etc.)
  - Enable domain specific languages
  - Leverage data-parallel execution engines from Intel
  - Provide interface specification for non-Intel implementations
  - Clearly specify semantics separately from syntax
  - Binary compatibility and insulation
- *Not* generally aimed at application developers

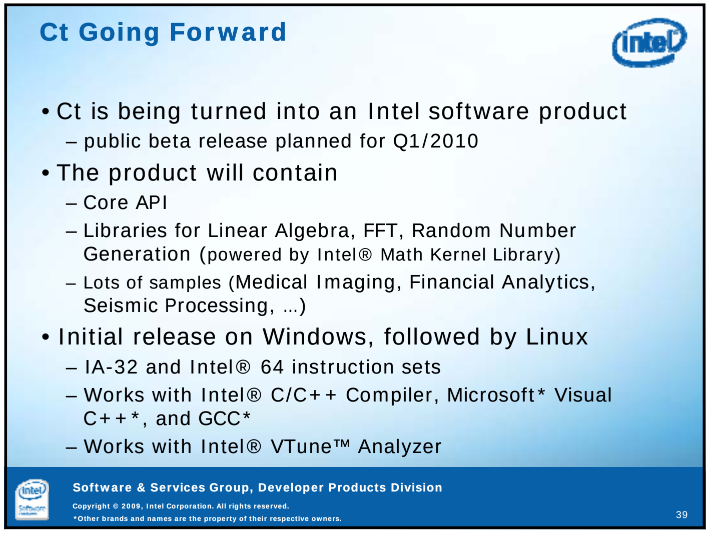
- Collaboration welcome!
  - Email to *stefanus.du.toit@intel.com*!

Software & Services Group, Developer Products Division



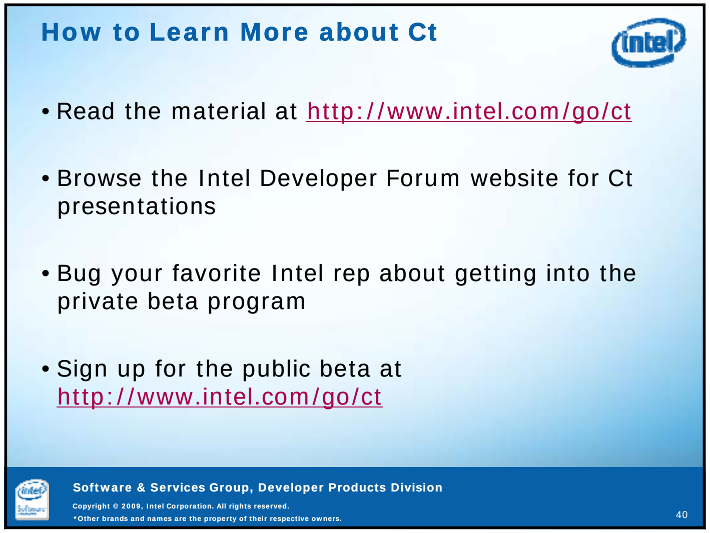
Ct In–Depth Information and Product Plans

Software & Services Group, Developer Products Division



Ct Going Forward

- Ct is being turned into an Intel software product
  - public beta release planned for Q1/2010
- The product will contain
  - Core API
  - Libraries for Linear Algebra, FFT, Random Number Generation (powered by Intel® Math Kernel Library)
  - Lots of samples (Medical Imaging, Financial Analytics, Seismic Processing, …)
- Initial release on Windows, followed by Linux
  - IA-32 and Intel® 64 instruction sets
  - Works with Intel® C/C++ Compiler, Microsoft* Visual C++*, and GCC*
  - Works with Intel® VTune™ Analyzer



How to Learn More about Ct

- Read the material at http://www.intel.com/go/ct
- Browse the Intel Developer Forum website for Ct presentations
- Bug your favorite Intel rep about getting into the private beta program
- Sign up for the public beta at http://www.intel.com/go/ct