Intel® Array Building Blocks

Productivity, Performance, and Portability with Intel® Parallel Building Blocks

Intel SW Products Workshop 2010
CERN openlab

Agenda

- Legal Information
- Optimization Notice
- Vision
- Call to Action
- Summary
- Contact
- Intel® Array Building Blocks
  - Introduction and Scope
  - Overview and workflow
  - Example: Mandelbrot set
  - Inside ArBB and tools
  - Performance and hints
- QnA

Legal Information

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPETY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Any code names and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user
- 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.
- Intel, Intel® Streaming SIMD Extensions (Intel® SSE), Intel® Advanced Vector Extensions (Intel® AVX), Intel® Parallel Building Blocks (Intel® PBB), Intel® Threading Building Blocks (Intel® TBB), Intel® Array Building Blocks (Intel® ArBB), Intel® Math Kernel Library (Intel® MKL), Intel® Integrated Performance Primitives (Intel® IPP), Intel® Cilk Plus and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
- Copyright ©2010 Intel Corporation.

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options". Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

Vision

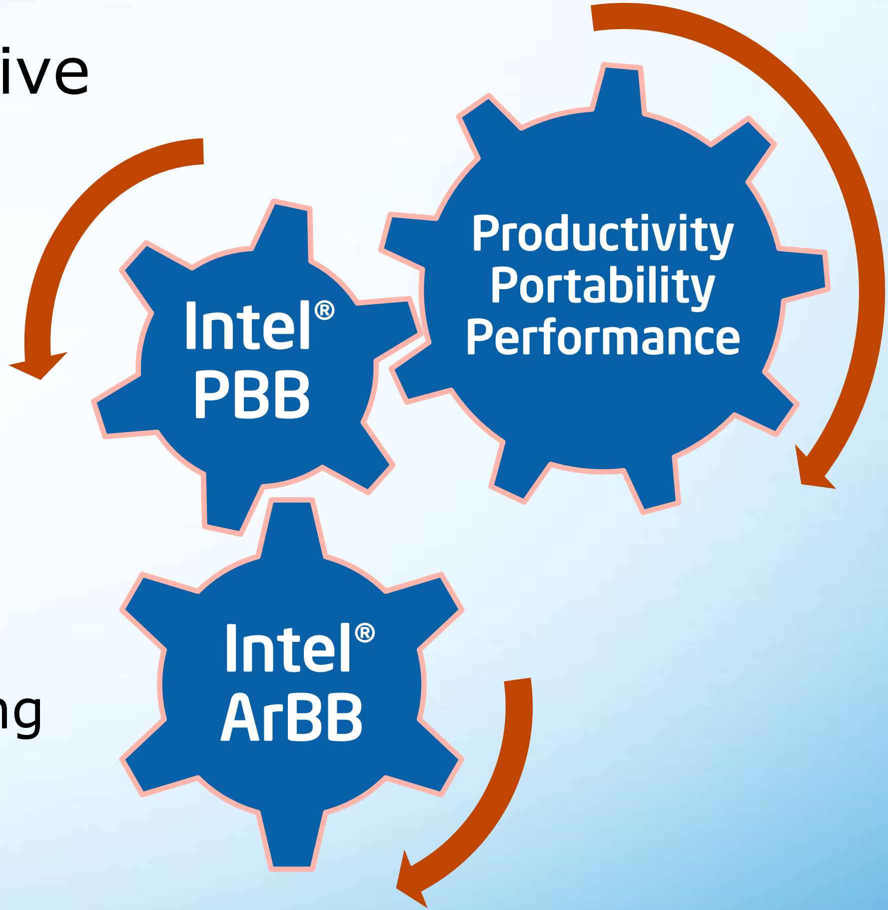
Intel ArBB is able to bring application programmers to parallel programming who would not address this domain otherwise.

- Improving the development workflow due to earlier introduction of parallelism
- Easier to add new hardware, and better performance experience
- Immediate advantage of next gen. hardware

Parallel programming is becoming the default

Call to Action

- Help to deliver an attractive programming model
  - Intel® ArBB becomes GOLD in May 2011
- Learn ArBB, or consider another member of the Parallel Building Blocks
  - Consider attending a training
  - Provide us your feedback



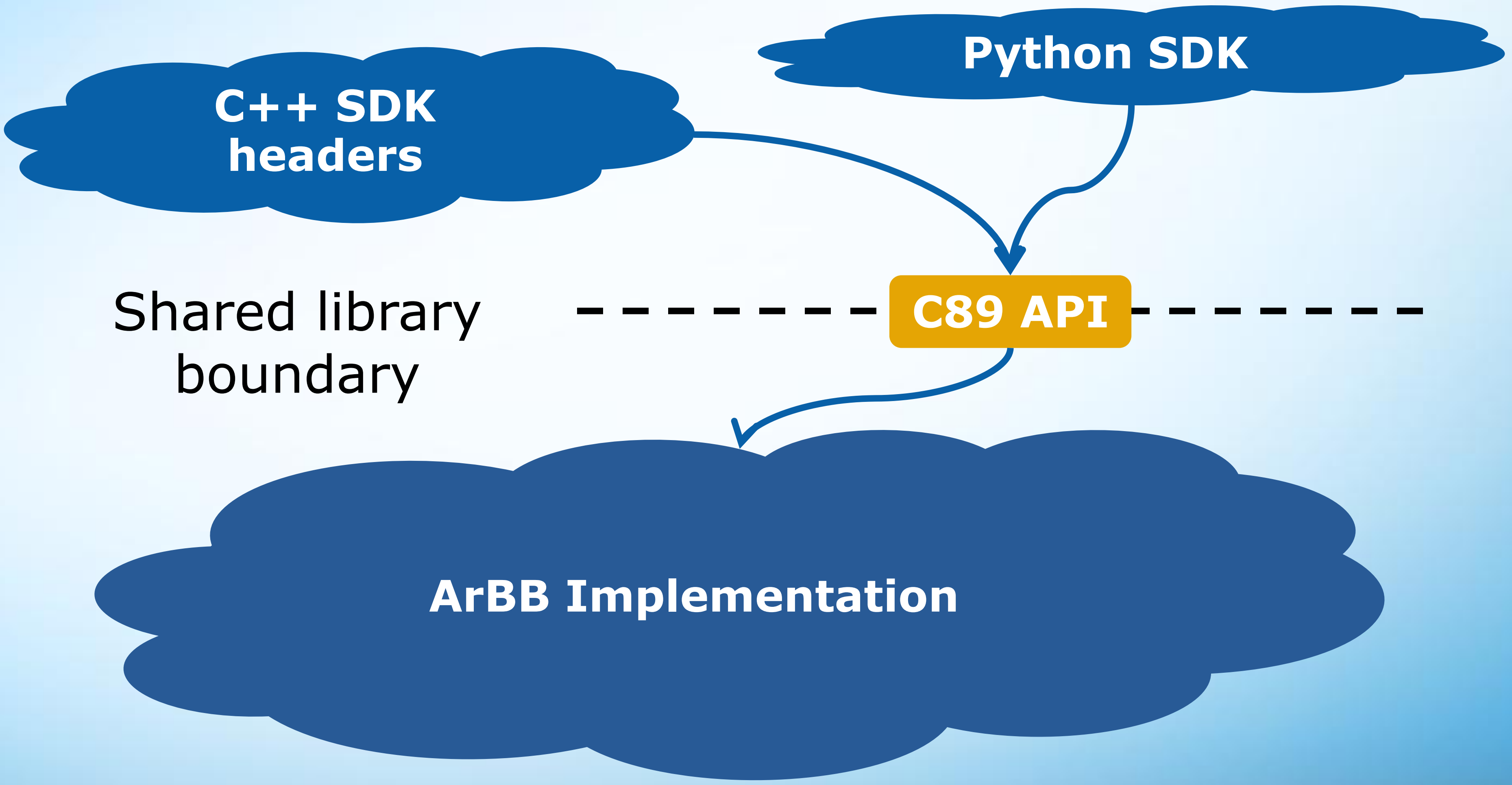
Intel's Family of Parallel Models

Intel® Parallel Building Blocks (PBB)		Fixed Function Libraries	Established Standards	Research and Exploration
Intel® Threading Building Blocks (TBB)	Intel® Cilk Plus	Intel® Math Kernel Library (MKL)	MPI	Intel® Concurrent Collections
Intel® Array Building Blocks (ArBB)		Intel® Integrated Performance Primitives (IPP)	OpenMP*	OpenCL*

Intel® Array Building Blocks (Intel® ArBB)

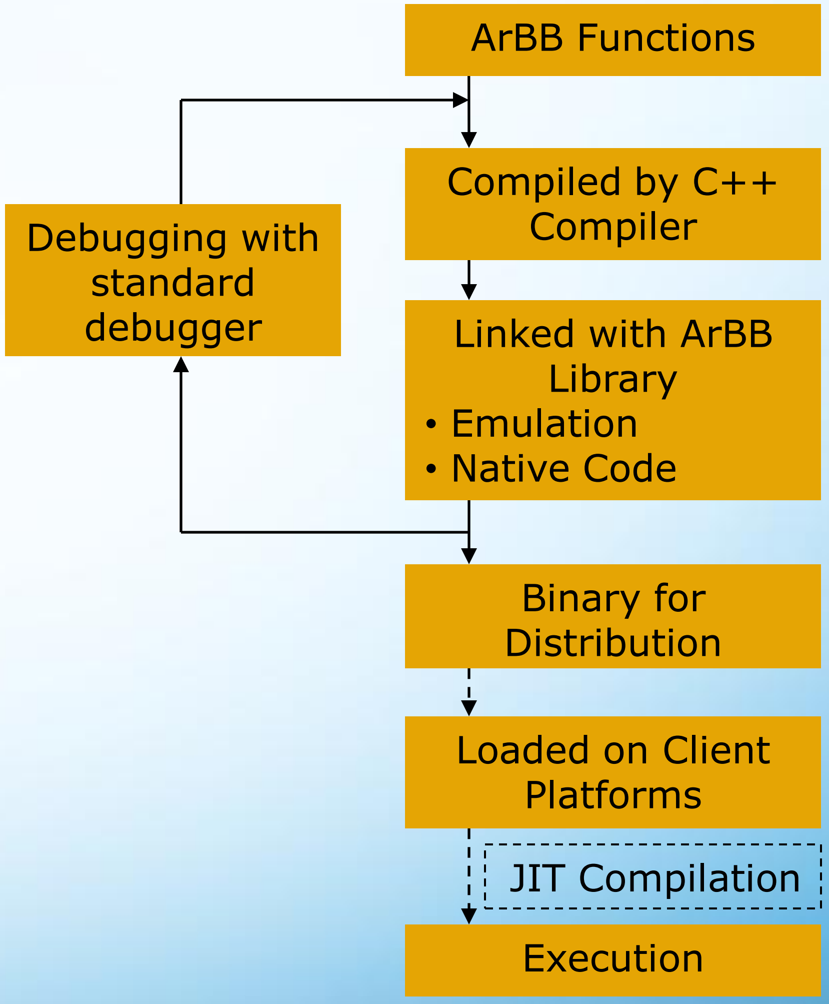
C++ library	• Embedded language • Dynamic compiler
Vector parallel	• Compute-intensive math • Implicit parallelism
Scalable	• Across multiple cores • Across varying SIMD width
Deterministic	• Structured data parallelism • No data races
Safety	• Separate memory space • No pointers

Language Binding



Library-based Approach and Workflow

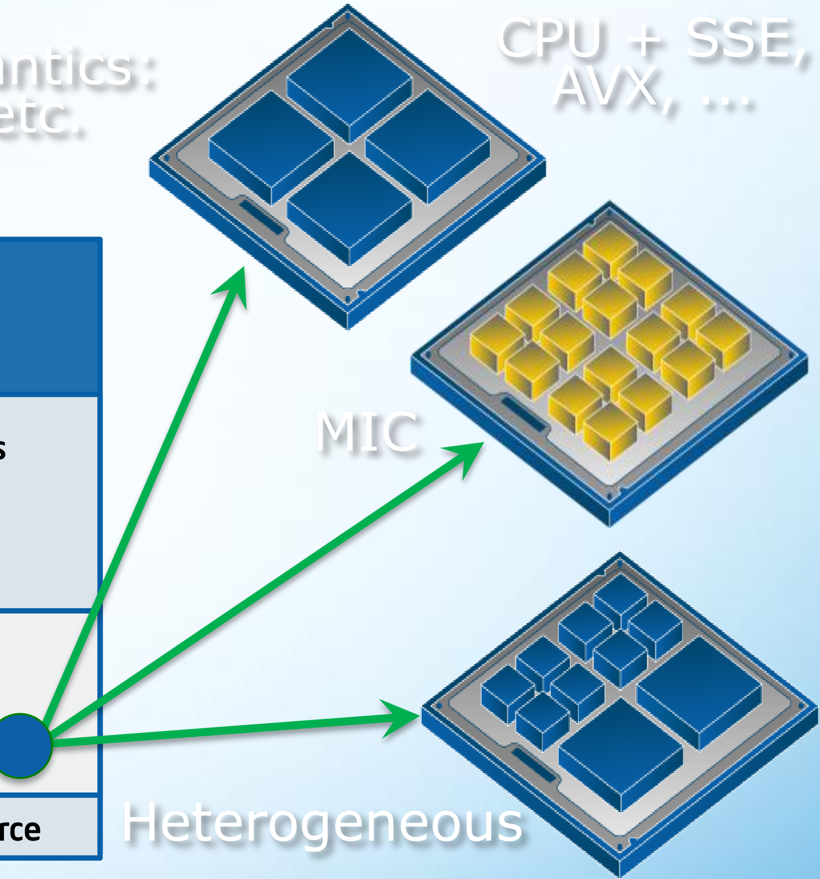
- Library-based approach
  - Virtual Machine C89-API
  - Mult. frontends possible
- C++ API-as-a-language
  - ISO-compliant C++ compiler is sufficient
  - C++ features available orthogonal to ArBB
  - Debugging using a standard debugger



Single Source, Multiple Targets

Sequentially consistent semantics:
no explicit threads, locks etc.

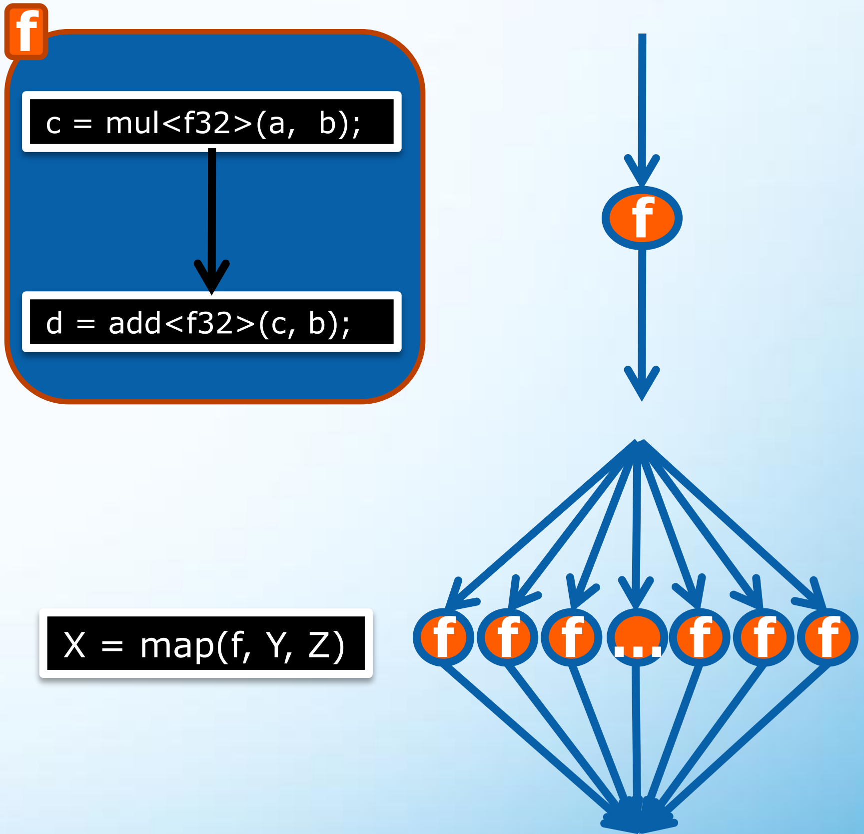
ArBB Functions in "serial" C++ App.	▪ Templates ▪ Overloaded operators
Standard C++ Compiler	▪ C++ ArBB statements are recorded into IR ▪ Links with dynamic library
ArBB Runtime	▪ Dynamic compiler ▪ Threading and heterogeneous runtime
	Single Source



Dynamically retargetable Execution

Sequentially Consistent Semantics

- Functions
  - Scoping complete inputs and outputs (no partitions)
  - Elemental functions (map) explicitly order-independent
- Operators
  - Parallelism is only inside
  - Arguments and outputs are logically “by-value”



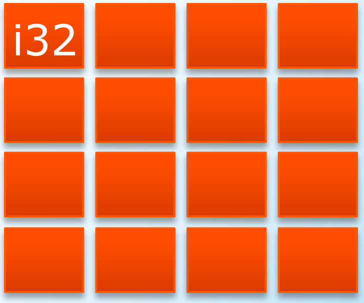
- Determinism feature
  - Same results regardless of utilizing multiple threads or single threaded execution (on the same machine)

Scalars and Containers

```
// dense vector of doubles
dense<f64> a("{0.0, 1.0, 2.0., 3.0}");
```

dense<T>

```
// dense matrix of integers
dense<i32, 2> b("{ { 0, 1, 2, 3 },
  { 4, 5, 6, 7 }, { 8, 9, 10, 11 } }");
// dense<T, N> with N := { 1, 2, 3}
```



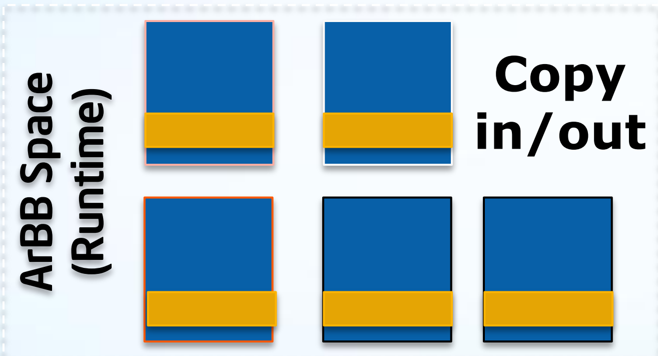
dense<T, 2>

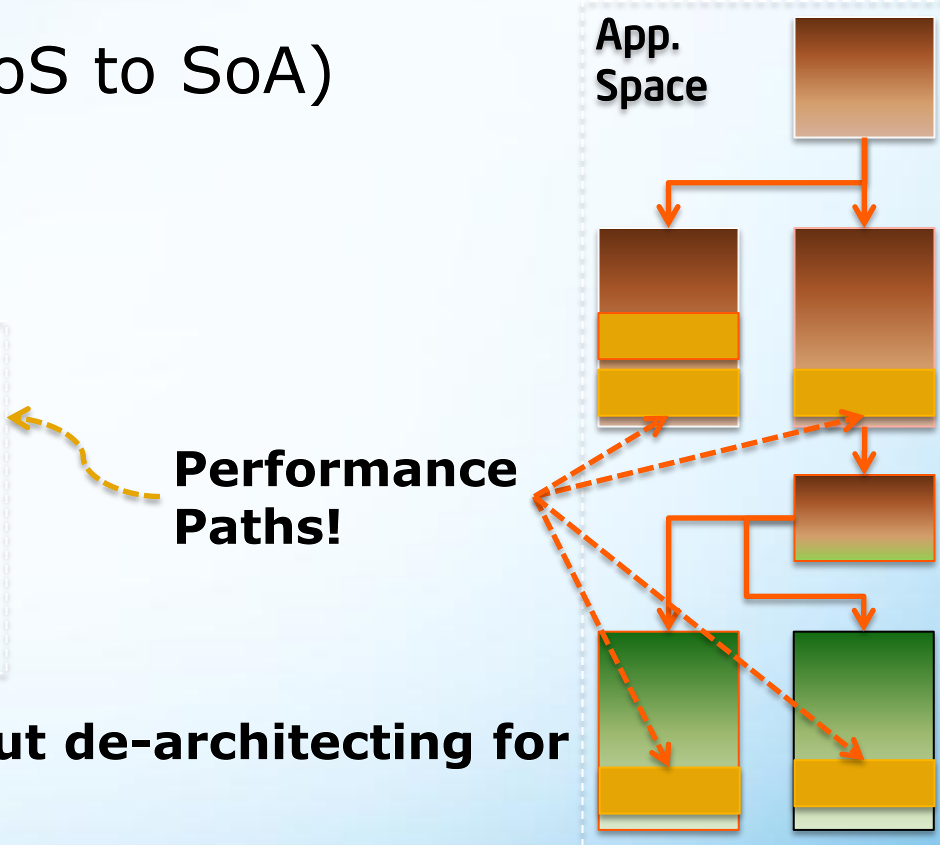
```
// irregular/segmented vector
nested<i32> b("{ { 0, 1 }, { 2, 3, 4 },
  { 5 }, { 6, 7, 8, 9 } }");
```

nested<T>

Segregated Memory Space

- Auto-vectorization (AoS to SoA)
- Remote-execution
- Safety





Optimized layout without de-architecting for performance

Safe by Default

Optimization Notice

Levels of Parallelism

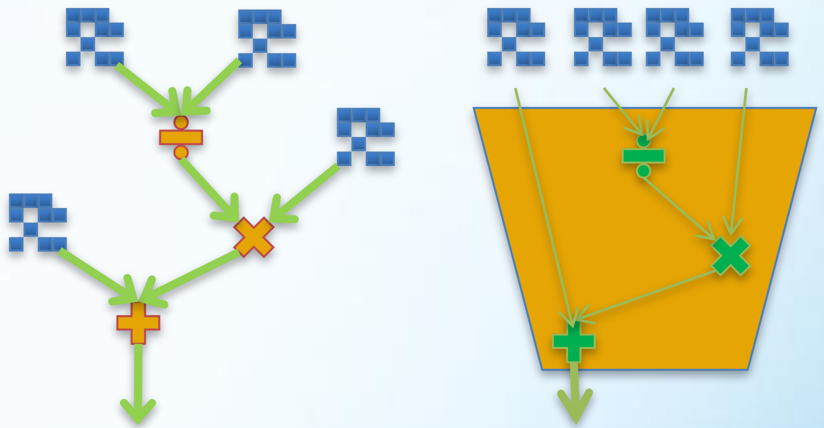
Grid	Group of clusters communicating through internet
Cluster	Group of computers communicating through fast interconnect
Node	Group of processors communicating through shared memory
Socket	Group of cores communicating through shared cache
Core	Group of functional units communicating through registers
Hyper-Threads	Group of thread contexts sharing functional units
Superscalar	Group of instructions sharing functional units
Pipeline	Sequence of instructions sharing functional units
Vector	Single instruction using multiple functional units

Parallelism and Fusion in ArBB

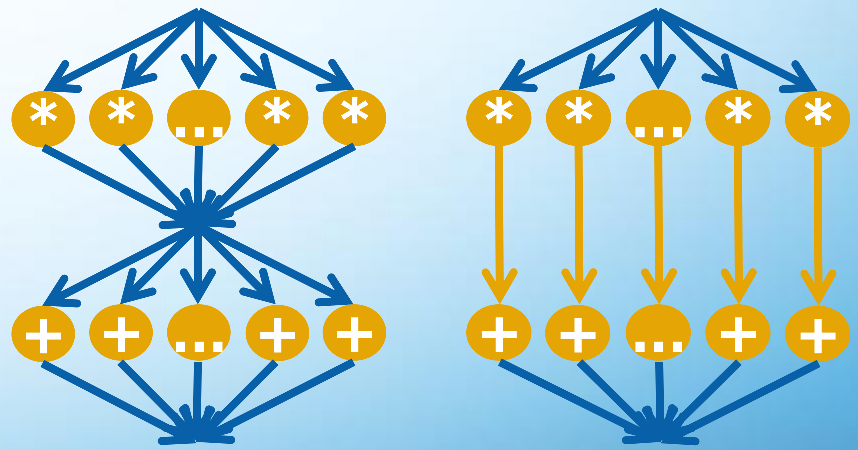
- Parallelism is implicitly expressed by containers
  - Idea: use containers and you are done to with parallelism

```
C = mul<dense<f32>>(A,  B);
```

```
D = add<dense<f32>>(C, B);
```



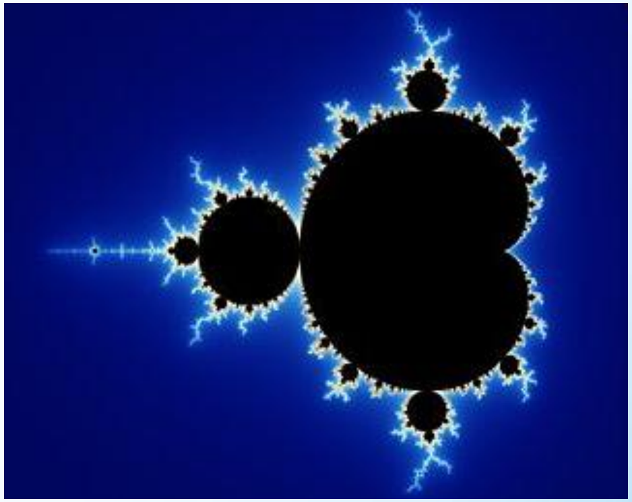
- Code is fused with respect to dependencies
  - Maximizing arith. intensity
  - Minimizing barriers (join+fork)



Mandelbrot Set

```
int max_count = 4711;
void mandel(i32& d, std::complex<f32> c) {
  i32 i;
  std::complex<f32> z = 0.0f;
  _for (i = 0, i < max_count, i++) {
    _if (abs(z) >= 2.0f) {
      _break;
    } _end_if;
    z = z * z + c;
  } _end_for;
  d = i;
}
void doit(dense<i32,2>& d, const dense<std::complex<f32>,2>& c)
{
  map(mandel)(d, c);
}

bind(pos, c_pos, cols, rows);
bind(dest, c_dest, cols, rows);
call(doit)(dest, pos);
```

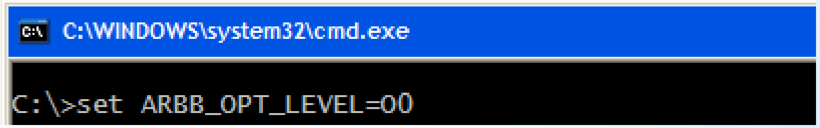


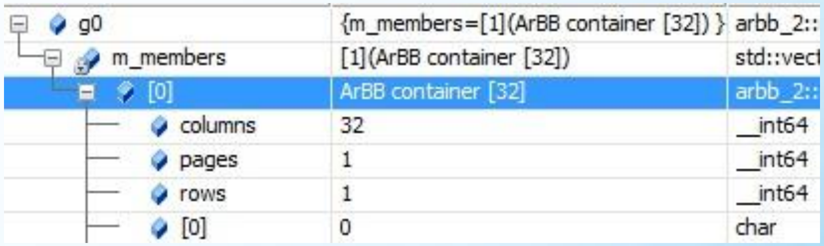
Color Legend:

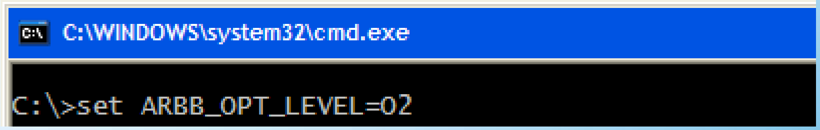
- ArBB Behavior
- ArBB Type

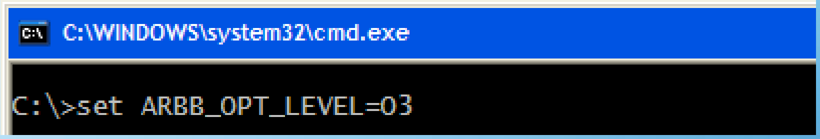
Workflow and Productivity

- Write correct algorithm first
  - Use immediate execution mode (no JIT code)
  - Work with approriate problem size (small)
  - Continue to use trusted tools, e.g. IDE, GDB etc.
  - No special debugger required ("debugger integration")
- Utilize SIMD and Threads
  - No additional effort (intent)
  - Tuning needed (real world)
  - Use performance analysis tools









Insight ArBB – "Opening the Black Box"

- Runtime Control
  - Environment options: during development
  - Mini API: for release and production
- Generate various dump files to get an overview
  - ARBB_VERBOSE = 1
  - ARBB_DUMPJIT = 1
- Tool integration: debugger, perf. analysis, ...

→ What's special about JIT-generated code?

JIT Code Generation by Keywords

Dynamic Compiler (JIT)

- Code becomes "data"
- Target-adaptive
- Optimization according to runtime state
- JIT symbol vs. source
- Compiler error appears at runtime (exception)

Static Compiler

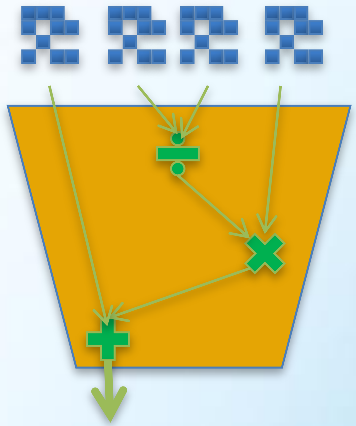
- Static "code path"
- Strong "proof"
- Compiler is separate from the application
- Linker: binaries (even libraries) and symbol format are standardized → mature tool support

Intel ArBB is including a Virtual Machine (VM) Specification and VM C89-API.

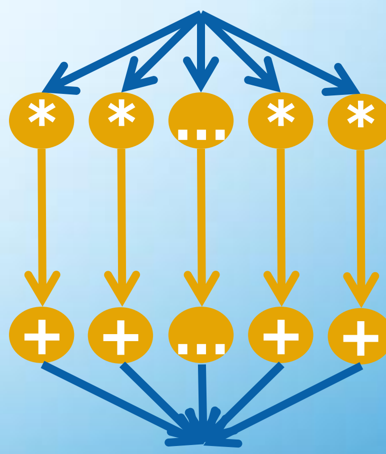
Performance and Operator Regularity

- Operator classification by regularity
  - Element-wise — very regular
  - Collective — mostly regular*
  - Permute — irregular
  - Misc. (facility) — depends

  * subject of barriers



- Fusion is increasing the work per task
  - Higher arithmetic intensity
  - Less task scheduling/threading overhead
  - Less synchronization overhead at barriers
  - Better locality (memory)



Performance Hints

- Prefer predefined operators over decomposition
  - Prefer API operators over syntax ("_for")
  - Keep container dimensionality ("upgrade")
- Understand exploiting runtime code generation
  - Application state can become a constant in JIT code
  - Possible to generate hard-coded "algorithms"
  - Compile-out OOP overhead
- Prefer GC memory rather than "binding"
- Predict JIT compilation using capture/closure

Select the appropriate PBB according to strengths.

Questions?

http://software.intel.com/en-us/articles/intel-array-building-blocks-documentation/
http://software.intel.com/en-us/articles/intel-array-building-blocks-kb/all/
http://software.intel.com/en-us/forums/intel-array-building-blocks/

Summary – ArBB

• Developer focus:	What to do, not how to do it
• Productivity:	Assume and debug as with serial programs
• Single source:	Multiple execution targets
• Optimization:	Fusion done by JIT compiler

- Syntax and semantics that extend C++
- Virtual Machine Specification and C API

Summary – PBB

Intel® Parallel Building Blocks

	Intel® Cilk Plus	Intel® Threading Building Blocks	Intel® Array Building Blocks
What it is	Language extensions to simplify parallelism	Widely used C++ template library for parallelism	Sophisticated C++ template library for data parallelism
Features	▪ 3 simple keywords ▪ Hyper-objects ▪ Array notations ▪ Sequential semantics ▪ Vectorization support	▪ Parallel Algorithms ▪ Data Structures ▪ Scalable Memory Allocator ▪ Task Scheduler ▪ Synchronization Primitives	▪ Uses both SIMD and multiple cores for data parallelism ▪ Safety guarantees to avoid data races and deadlocks ▪ Vectorization support
Reasons to Use	▪ Simplest way to parallelize your code ▪ Serial semantics + low overhead = powerful solution ▪ Supports C & C++; Windows* and Linux*	▪ Rich feature set for general purpose parallelism ▪ Available as open source or commercial ▪ Supports C++; Windows, Linux, Mac OS*, other OS's	▪ Sophisticated data parallel support includes vectorization, dense, sparse and irregular matrix support ▪ JIT & VM technology = flexible and powerful ▪ Supports C++; Windows & Linux

MIX AND MATCH TO OPTIMIZE YOUR APPLICATION'S PERFORMANCE

Contact

Hans Pabst, Software Engineer TCE ***hans.pabst@intel.com***
Performance and Productivity Libraries

11/29/2010 27