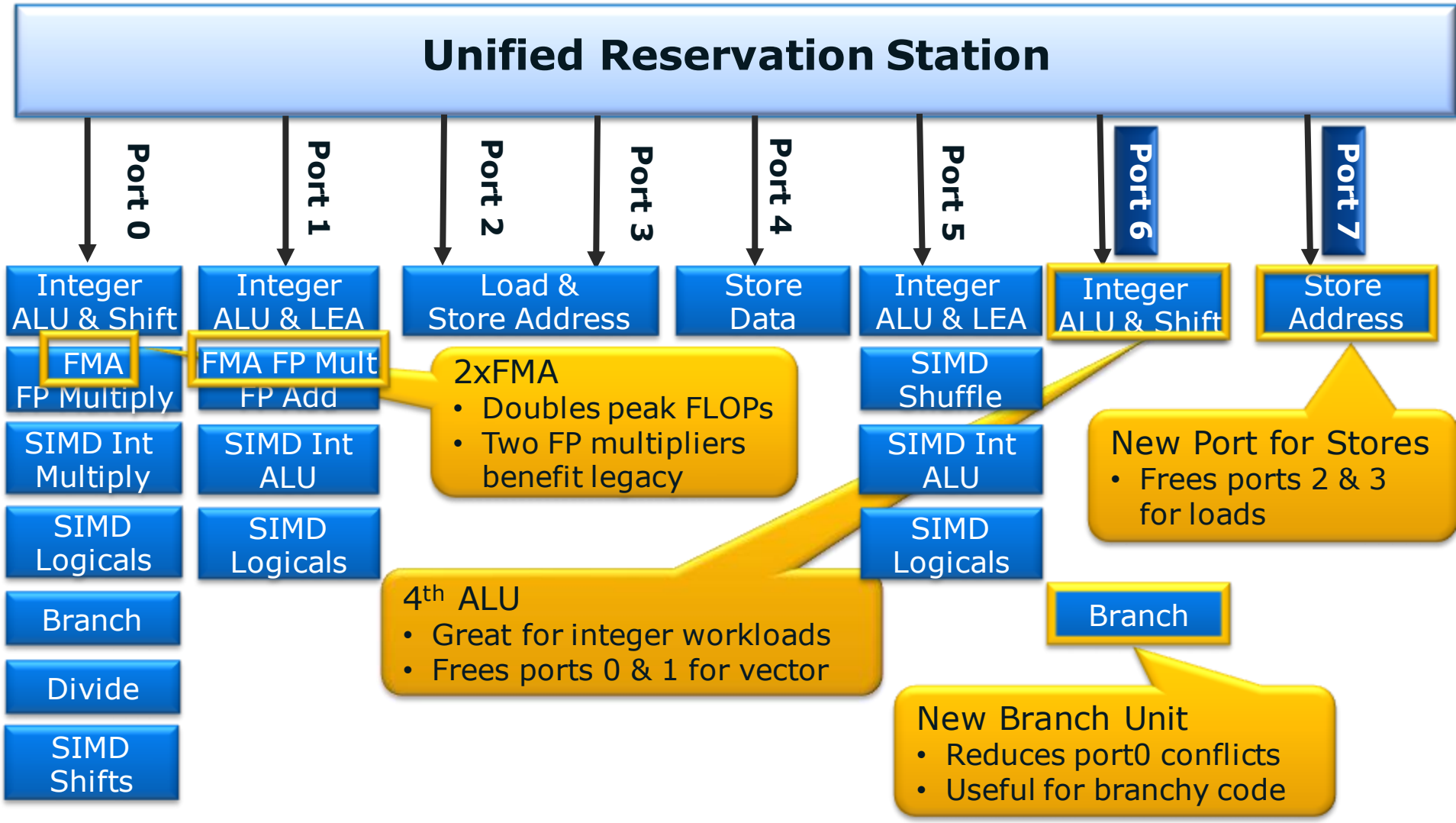
Getting the most from Haswell

(4th Generation Intel® Core™ microarchitecture)

Arch D. Robison

Sr. Principal Engineer, Intel

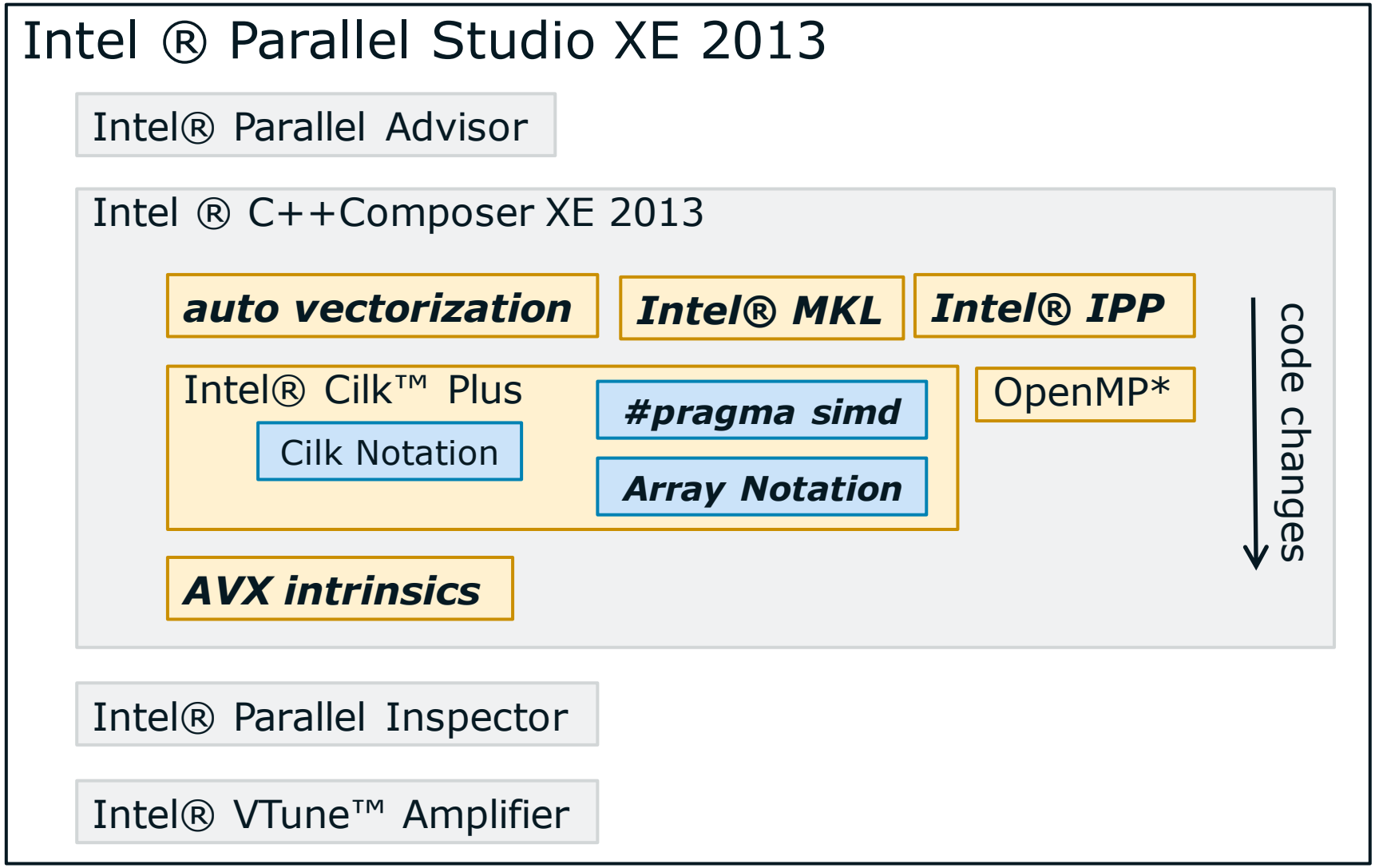
Haswell Execution Unit Overview



New Instructions in Haswell

Group		Description
AVX2	SIMD Integer Instructions promoted to 256 bits	Extend vector integer instructions to 256 bit.
	Gather	Load elements from vector of indices
	Shuffling and Data Rearrangement	Blend, shift, and permute instructions
FMA		Fused Multiply-Add
Bit Manipulation and Cryptography		Improve performance of bit-stream manipulation and decode, large integer arithmetic, and hashes
TSX=RTM+HLE		Transactional Memory
Others		MOVBE: Load and store big-endian data INVPCID: Invalidate processor context id

Leveraging Vector Instructions



New Operations in Intel® AVX2

New Instruction	Description
VPERM2I128 VEXTRACTI128 VINSERTI128 VPMASKMOV{D,Q}	Integer versions of Intel® AVX cross lane permutes & masked load/stores
VPERM{Q,PD} VPERM{D,PS}	256-bit "Cross-lane" Permutes
VPSLLV{D,Q} VPSRLV{D,Q} VPSRAVD	Per Element Variable Vector Shifts
VPBLENDD	Element blend
VPBROADCAST{B,W,D,Q} VBROADCAST{SS,SD}	New Broadcasts Include register-to-register broadcasts

Intel AVX2 introduces 20+ new operations to already rich vector instruction set, plus Gather & FMA

High-Level Programming

Array Notation

```
void foo( int c[], int b[], int a[], int n ) {
    c[0:n] = b[0:n] << a[0:n];
}
```

OpenMP 4.0 simd

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

Intel® Compiler 14.0 recommended icc -xCORE-AVX2 -O3 -openmp

New Gather Instructions

Gather = Vector load: A[B[i]]

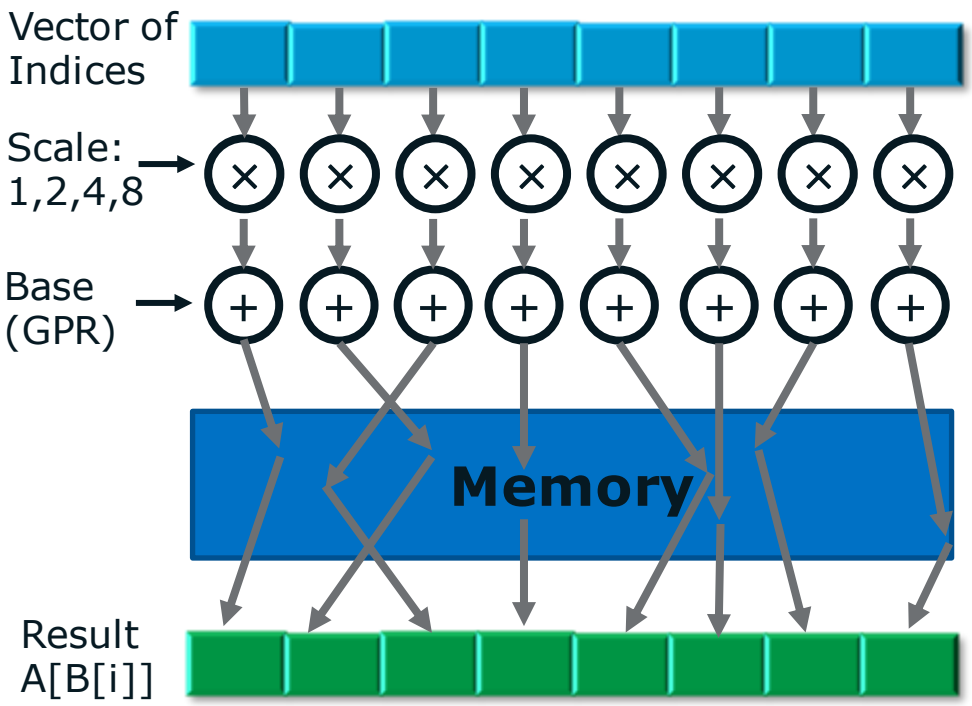
- [Index]×Scale + Base
- 32b or 64b elements
- 32b or 64b indices

Mask: Elements to be gathered

- Complete when Mask=0

Should be used with caution depending on

- Value of mask
- Index reuse
- Number of elements



Fundamental building block for sparse, indirect memory accesses, easing vectorization

High-Level Programming

Array Notation

```
void bar( float c[], float b[], float a[], int j[], int n ) {
    c[0:n] += a[j[0:n]]*b[0:n];
}
```

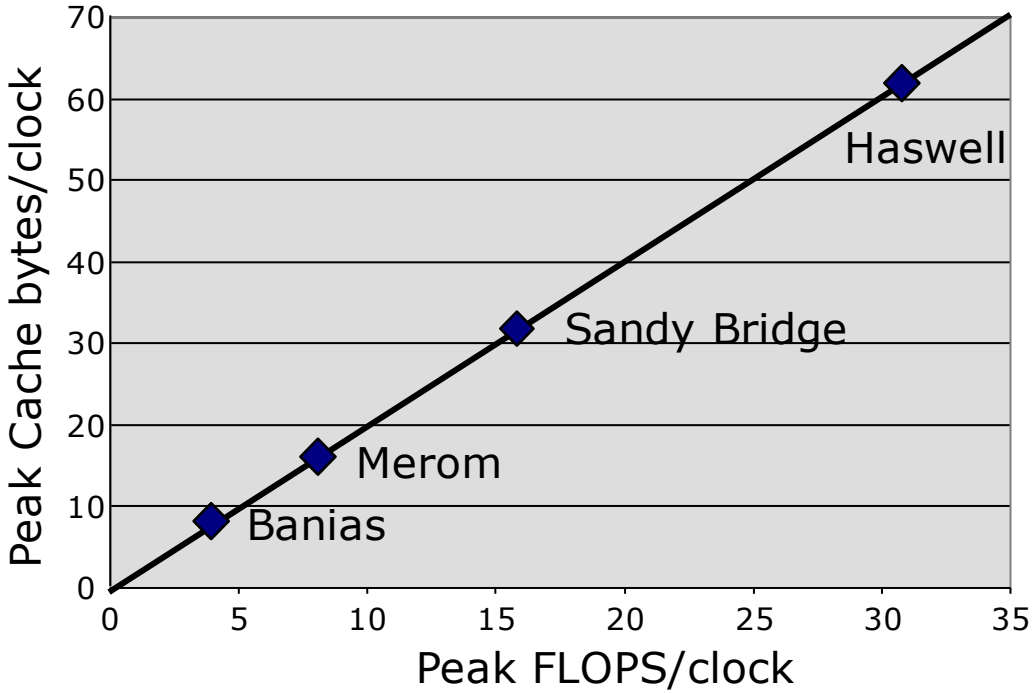
OpenMP 4.0 simd

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

FMA: Fused Multiply-Add

- Computes **±(a×b)±c** with only one rounding
  - Increased accuracy compared to MUL & ADD
  - Latency same as for **×** alone.
- 2xFMA units → Double peak FLOPs
  - **Doubled cache bandwidth** to feed FMA

FMA provides improved accuracy and performance



All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

High-Level Programming

Array Notation

```
void bar( float c[], float b[], float a[], int j[], int n ) {
   c[0:n] += a[j[0:n]]*b[0:n];
}
```

OpenMP 4.0 simd

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

FMA useful for scalar code too.

SIMD (Elemental) Functions

Annotation for creating clones suitable for vectorized call sites.

Callee

```
#pragma omp declare simd linear(a), uniform(b)
void bar( float* a, float* b, int j ) {
    if( *a>0 )
       *a = b[j];
}
```

Caller

```
__declspec(vector(linear(a),uniform(b)))
void bar(float *a, float *b, int c, int d);

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

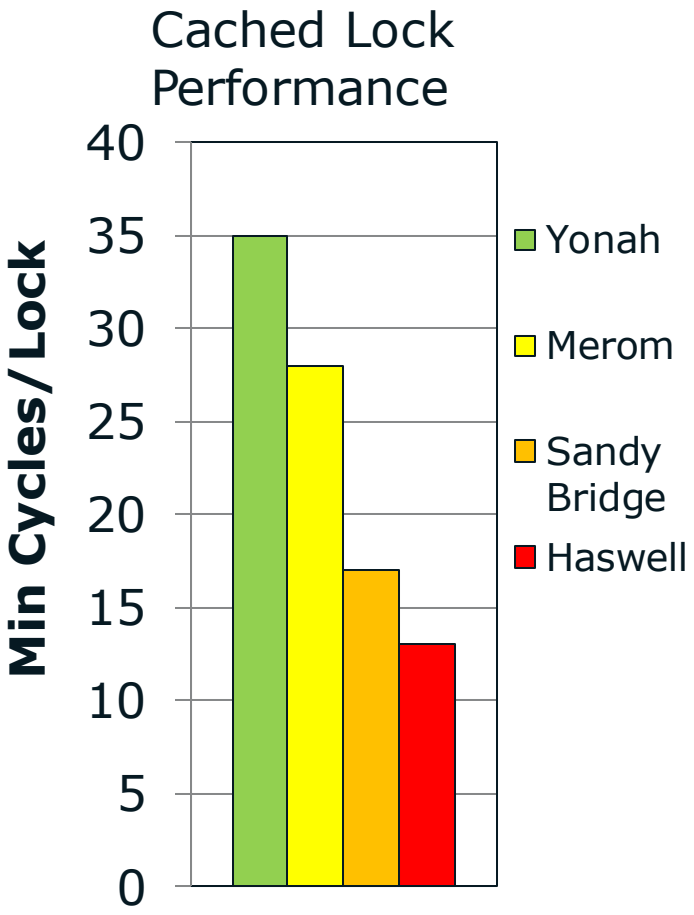
New Bit Manipulation Instructions (BMI)

Category	Name	Operation	
Bit-field manipulation	BZHI	Zero High Bits Starting with Specified Position	
	SHLX	New Variable Shifts (non-destructive, have load+operation forms, no implicit CL dependency, and no flags effect)	x<<y
	SHRX		x>>y
	SARX		x>>y
	BEXTR	Bit Field Extract using a pair of parameters in a single register (use BZHI/SHRX combo for inputs in independent regs though)	
	PDEP	Parallel Bit Deposit (bit scatter)	
	PEXT	Parallel Bit Extract (bit gather)	
	ANDN	Logical And-Not	~x&y
Leading and trailing zero bit counts; Trailing set bit manipulation	TZCNT	Trailing Zero Bits Count	
	LZCNT	Leading Zero Bits Count	
	BLSR	Reset Lowest Set Bit	x-1 & x
	BLSMSK	Get Mask Up to Lowest Set Bit	x-1 ^ x
	BLSI	Isolate Lowest Set Bit	-x & x
Improved multiply and rotate	RORX	Rotate Without Affecting Flags	
	MULX	Unsigned Multiply Without Affecting Flags	

15 new GPR Instructions to accelerate bit manipulation and compression routines

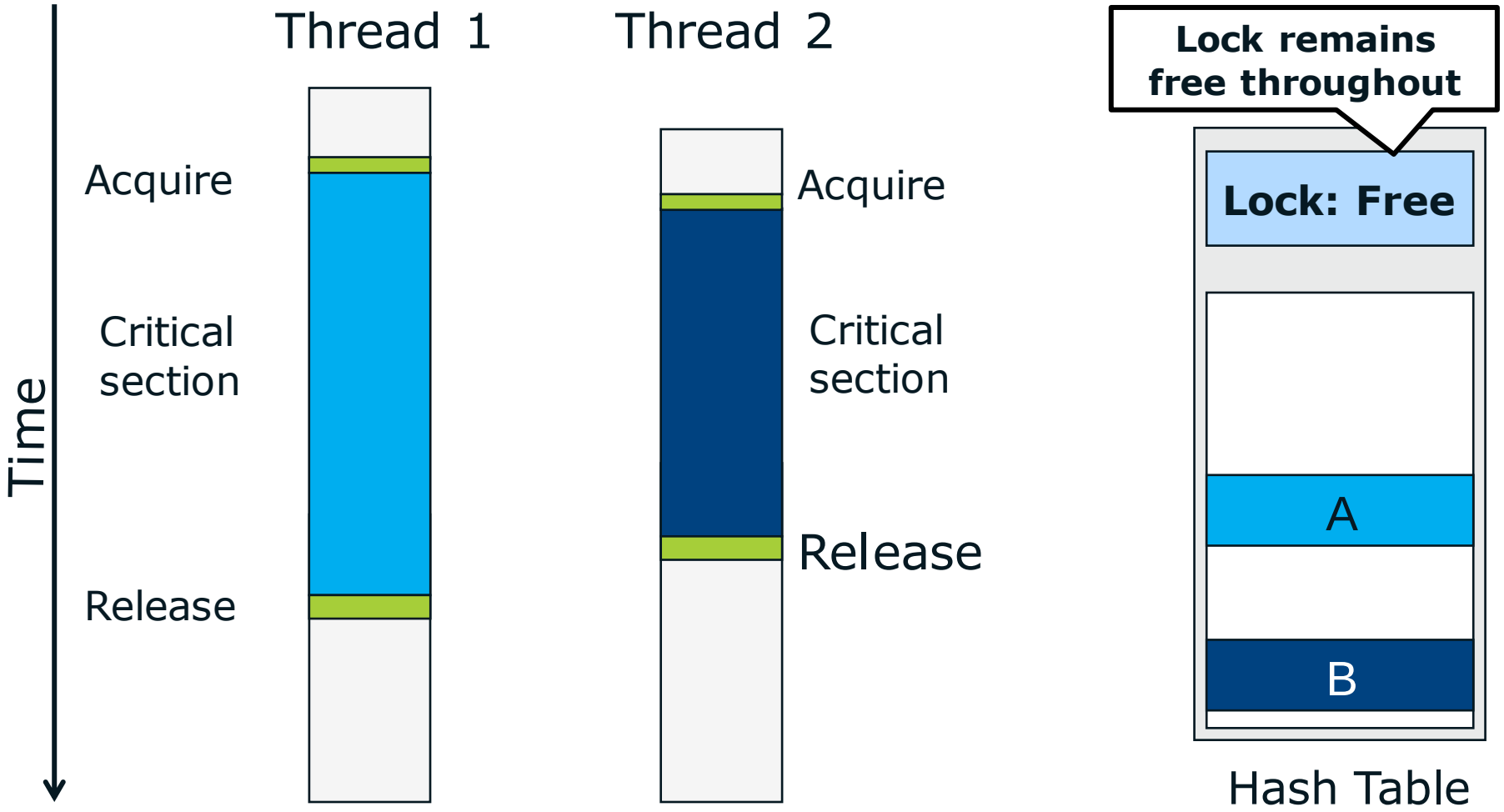
Synchronization Improvements

- **Improving existing primitives**
  - Faster LOCK-prefixed instructions
  - A focus in recent generations
- **Locks still limit concurrency**
  - Fine-grained locking can help.
  - But is tricky to get right.
- **Intel® TSX**
  - Enables concurrent execution of **transactions** that do not conflict.
  - Performance of fine-grained locks with effort of coarse-grained locks



Intel® TSX exposes parallelism through Transactions

Example Intel® TSX Execution



No serialization and no communication if no conflicts

Two interfaces to Intel® TSX

Hardware Lock Elision (HLE)

- Legacy-compatible approach
- Looks like plain locks to old hardware

Restricted Transactional Memory (RTM)

- Flexible interface for creating your own transactions
- New instructions – not backwards compatible

Hardware Lock Elision (HLE)

New instruction prefixes

- **XACQUIRE prefix on instruction that locks**
- **XRELEASE** prefix on instruction that unlocks

Hardware without TSX:

- prefixes ignored

Hardware with TSX:

- Tries to execute critical section transactionally
- Concurrency possible if there is no data conflict
- Abort causes a re-execution with a lock

Useful for Exploiting Transactions in Backwards Compatible Way

Restricted Transactional Memory (RTM)

XBEGIN *fallback*	Start transaction
XEND	End transaction
XABORT *abortcode*	Abort transaction
XTEST	Inside a transaction?

There is no guarantee that a transaction will ever succeed!

- Must provide a fallback path (usually a real lock)

Read Chapter 12 of the latest *Intel® 64 and IA-32 Architectures Optimization Reference Manual* for advice on using Intel® TSX.

Usage from C/C++ using Intrinsics

```
retry:
    if( _xbegin()==~0u) {
        ... critical section...
        if( another thread acquired the lock )
            _xabort(errorcode)
        else
            _xend();
    } else {
        wait until lock is free;
        if(seems worth another try)
            goto retry;
        acquire lock
        ... critical section ...
        release lock
    }
```

rollback

transactional execution

avoid "Lemming Effect"

avoid "Groundhog Day"

Need icc ≥12.1
or gcc ≥ 4.8
or Visual Studio ≥ 2012

Library Support for Lock Elision

Intel® Threading Building Blocks (Intel® TBB)

```
tbb::speculative_spin_mutex  mutex;
```

```
void foo()
   ...
   {
       tbb::speculative_spin_mutex::scoped_lock  lock(mutex);
       ...critical section...
       // Destructor implicitly releases lock
    }
    ...
}
```

Summary

Haswell cores are wide

- 256-bit vector units
- Lots of implicit instruction-level parallelism

Intel® Transactional Synchronization Extensions (Intel® TSX)

- New programming paradigm

Expressing parallelism is important!

- SIMD
  - OpenMP* 4.0 simd
  - Array Notation
- Threading
  - OpenMP*
  - Intel® Cilk™ Plus
  - Intel® Threading Building Blocks

Resources:

Intel® Parallel Studio XE Suites

http://software.intel.com/en-us/intel-parallel-studio-xe

Intel® AVX1, AVX2, BMI and TSX:

http://www.intel.com/sdm

Intel® Advanced Vector Extensions 512 (Intel® AVX-512) and beyond:

http://www.intel.com/software/isa

Discussion forum:

http://software.intel.com/en-us/forums/intel-isa-extensions

Intel® Software Developer Emulator (SDE)

Emulate new instructions before hardware is available

http://www.intel.com/software/sde

Intel® Architecture Code Analyzer

http://software.intel.com/en-us/articles/intel-architecture-code-analyzer

Core Cache Size/Latency/Bandwidth

Metric	Nehalem	Sandy Bridge	Haswell
L1 Instruction Cache	32K, 4-way	32K, 8-way	32K, 8-way
L1 Data Cache	32K, 8-way	32K, 8-way	32K, 8-way
Fastest Load-to-use	4 cycles	4 cycles	4 cycles
Load bandwidth	16 Bytes/cycle	32 Bytes/cycle (banked)	**64 Bytes/cycle**
Store bandwidth	16 Bytes/cycle	16 Bytes/cycle	**32 Bytes/cycle**
L2 Unified Cache	256K, 8-way	256K, 8-way	256K, 8-way
Fastest load-to-use	10 cycles	11 cycles	11 cycles
Bandwidth to L1	32 Bytes/cycle	32 Bytes/cycle	**64 Bytes/cycle**
L1 Instruction TLB	4K: 128, 4-way 2M/4M: 7/thread	4K: 128, 4-way 2M/4M: 8/thread	4K: 128, 4-way 2M/4M: 8/thread
L1 Data TLB	4K: 64, 4-way 2M/4M: 32, 4-way 1G: fractured	4K: 64, 4-way 2M/4M: 32, 4-way 1G: 4, 4-way	4K: 64, 4-way 2M/4M: 32, 4-way 1G: 4, 4-way
L2 Unified TLB	4K: 512, 4-way	4K: 512, 4-way	**4K+2M shared: 1024, 8-way**

All caches use 64-byte lines

Intel® microarchitecture code names Nehalem, Sandy Bridge, and Haswell

Legal Disclaimer & Optimization Notice

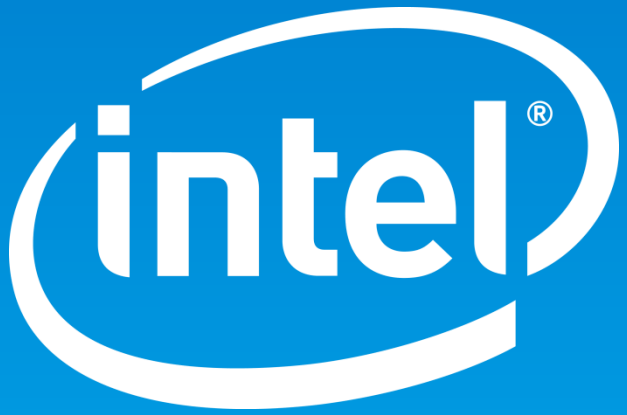
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING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