Intel® Xeon Phi™ Programmability (the good, the bad and the ugly)

Robert Geva
Parallel Programming Models Architect
robert.geva@intel.com
My Perspective

• When the compiler can solve the problem

• When the programmer has to solve the problem

• When expectations mismatch
Iterative HW investments

PARALLEL

PROGRAMMABLE

robert.geva@intel.com
Both parallel and programmable
4 C++ based programming systems
4 C++ based programming systems
4 C++ based programming systems
4 C++ based programming systems
Heterogeneous Programming for Xeon Phi

Offload Extensions

Programming the coprocessor is the same as programming the CPU
Heterogeneous Programming for Xeon Phi

Same parallelization techniques apply

Offload Extensions

MKL
TBB
OpenMP
Cilk Plus
C++/FTN
OS

Parallel Compute

CPU Executable

MIC Native Executable

Parallel Compute

PCIe

Heterogeneous Compute
Parallel Programming for Intel® Architecture

**Cores**
- Use threads, directly or via OpenMP*, or
- Use tasking, Intel® TBB / Cilk™ Plus

**Vectors**
- Intrinsics, auto vectorization
- **Language extensions for vector programming**

**Blocking algorithms**
- Use caches to hide memory latency
- Organize memory access for data reuse

**Data layout and alignment**
- Structure of arrays facilitates vector loads / stores, unit stride
- Align data for vector accesses
Shared Memory Parallelism for IA

- **OpenMP***
  - Well known industry standard
  - Best suited when resource utilization is known at design time

- **Intel® TBB**
  - C++ Library of parallel algorithms, containers
  - Load balancing via work stealing

- **Intel® Cilk™ Plus**
  - Serial equivalence via compiler
  - Load balancing via work stealing

Different choices for different uses
Cache Efficient Algorithms

• Blocking / Tiling
  – Needs some retuning per platform

• Divide and conquer algorithms
  – Tend to use recursion and parallelize it

• Cache oblivious algorithms
  – Expected to provide asymptotically optimal cache miss behavior
  – In a target independent design
  – Very hard to write

• Is all the burden on falling on the programmer?
Cache Efficient Stencils?

for (t = 1; t≤T, ++t) {
    for (i0 = 0, i0<n0, ++i0) {
        for (i1 = 0, i1<n1, ++i1) {
            for (i2 = 0, i2<n2, ++i2) {
                update A[t%k,i0,i1,i2] according to stencil
            }
        }
    }
}

Looping is memory intensive, especially for parallel implementations, and it uses the caches poorly. Assuming data-set size N, cache-block size B, cache size M < N, the number of cache misses is Θ(N/B).
Cache-Oblivious Stencil Algorithms

Divide-and-conquer cache-oblivious techniques, based on *trapezoidal decompositions* [FrigoSt05], are known to be effective. **Problem:** These codes are difficult to write.

- **Pochoir:** a functional stencil language embedded in C++
- The programmer just provides the stencil code and data, writing serial code
- Pochoir employs a novel cache-oblivious algorithm for arbitrary d-dimensional grids
- Allows arbitrary periodic and nonperiodic boundary conditions
- Implements a variety of stencil-specific optimizations.

Robert Geva
robert.geva@intel.com

Data Layout
Structured data is common, e.g.

But an array of these looks like this:

![Data Layout Diagram]

Inefficient! To gather all the reds, for instance, we must gather

![4 loads!]

What we need is SOA layout.

![1 load!]
AOS $\rightarrow$ SOA Conversion

- **Manual**: The programmer carries the burden
- **Feedback**:  
  - Layout the data is acceptable  
  - writing SOA code is not.

```cpp
for (auto i=y0; i<len; ++i) {
    arr.x[i] += arr. [i] * arr.z[i];
    result += arr.x[i];
}
```

- **Direction (work in progress)**:  
  - Provide a C++11 based conversion library  
  - Allow the programmer to write AOS, object oriented syntax  
  - It gets converted to SOA code, accessing SOA data layout  
  - The syntactic overhead is optimized away  
  - Same performance as manual solution
Vector Parallelism in Intel® Cilk™ Plus

Array Notations
- Syntax to operate on arrays
- No ordering constraints → use SIMD

Elemental Functions
- Function describes operations on an element
- Deployed across a collection of elements

SIMD Loops
- Vector parallelism on a single thread
- Guaranteed vector implementation by the compiler

Language support for explicit vector programming
Example: Monte Carlo

```c
#pragma omp parallel for
for(int opt = 0; opt < OPT_N; opt++)
{
    float VBySqrtT = VOLATILITY * sqrtf(T[opt]);
    float MuByT = (RISKFREE - 0.5f * VOLATILITY * VOLATILITY) * T[opt];
    float Sval = S[opt];
    float Xval = X[opt];
    float val = 0.0f, val2 = 0.0f;
    #pragma simd reduction(+:val) reduction(+:val2)
    for(int pos = 0; pos < RAND_N; pos++){
        float callValue = expectedCall(Sval, Xval, MuByT, VBySqrtT, l_Random[pos]);
        val += callValue;
        val2 += callValue * callValue;
    }
    float exprt = expf(-RISKFREE * T[opt]);
    h_CallResult[opt] = exprt * val / (float)RAND_N;
    float stdDev = sqrtf((((float)RAND_N*val2 - val*val) / ((float)RAND_N*(float)(RAND_N - 1.f))));
    h_CallConfidence[opt] = (float)(exprt * 1.96f * stdDev/sqrtf((float)RAND_N));
}
```
The Same Source Change Improves Performance on Both platforms

Parallelization and vectorization together improve options per second by > 800X and by >50X
Outer Loop Vectorization

```c
#pragma simd
for (i=0; i<n; i++) {
    complex<float> c = a[i];
    complex<float> z = c;
    int j = 0;
    while ((j < 255) && (abs(z)< limit)) {
        z = z*z + c;
        j++;
    }
    color[i] = j;
}
```
Data in Vector Loops

- The two statements with the += operations have different meaning from each other
- The programmer should be able to express those differently
- The compiler has to generate different code
- The variables $i$, $p$ and $step$ have different “meaning” from each other

```c
float sum = 0.0f;
float *p = a;
int step = 4;
#pragma simd reduction(+:sum) linear(p:step)
for (int i = 0; i < N; ++i) {
    sum += *p;
    p += step;
}
```
Heterogeneous Example: Computing Pi

```c
#define NSET 1000000
int main ( int argc, const char** argv )
{
    long int i;
    float num_inside, Pi;
    num_inside = 0.0f;
#pragma offload target (MIC)
#pragma omp parallel for reduction(+:num_inside)
    for( i = 0; i < NSET; i++ )
    {
        float x, y, distance_from_zero;
        // Generate x, y random numbers in [0,1)
        x = float(rand()) / float(RAND_MAX + 1);
        y = float(rand()) / float(RAND_MAX + 1);
        distance_from_zero = sqrt(x*x + y*y);
        if ( distance_from_zero <= 1.0f )
            num_inside += 1.0f;
    }
    Pi = 4.0f * ( num_inside / NSET );
    printf("Value of Pi = %f \n",Pi);
}```

A one line change from the CPU version
Offloading “a kernel”

```c
__declspec (vector)
double option_price_call_black_scholes(
    double S,       // spot (underlying) price
    double K,       // strike (exercise) price,
    double r,       // interest rate
    double sigma,   // volatility
    double time)    // time to maturity
{
    double time_sqrt = sqrt(time);
    double d1 = (log(S/K)+r*time)/(sigma*time_sqrt)+0.5*sigma*time_sqrt;
    double d2 = d1-(sigma*time_sqrt);
    return S*N(d1) - K*exp(-r*time)*N(d2);
}
```

//offload. Data is in lexical scope, the compiler copies it
#pragma offload target(MIC)
#pragma omp parallel for
for (int i=0; i<NUM_OPTIONS; i++) {
    call[i] = option_price_call_black_scholes(S[i], K[i], r, sigma, time[i]);
}
```
You can offload anything

//parallelism structure does NOT have to be known at offload point
#pragma offload target(MIC) in(my_data) out(my_result)
a_third_party_function();

#pragma offload target(mic) in(my_data) out(my_result)
#pragma omp parallel for
for (i=0; i<N; ++i) {
    for (j = 0; j<i; ++j) {
        #pragma omp parallel for {
            for (k=0; k<M;++k) {
                body(i,j,k);
            }
        }
    }
}
Parallel Programming for Intel® Architecture

- **Cores**
  - Use threads, directly or via OpenMP®, or
  - Use tasking, Intel® TBB / Cilk™ Plus

- **Vectors**
  - Intrinsics, auto vectorization
  - Language extensions for vector programming

- **Blocking algorithms**
  - Use caches to hide memory latency
  - Organize memory access for data reuse

- **Data layout and alignment**
  - Structure of arrays facilitates vector loads / stores, unit stride
  - Align data for vector accesses
THANK YOU!
Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY 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.

• A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENCE IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

• Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

• The products described in this document 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.

• Intel product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel's current plan of record product roadmaps.

• Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number.

• Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

• Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

• IVB, KNC, MIC 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.

• Intel, Xeon, Xeon Phi, Cilk , Ultrabook, Sponsors of Tomorrow 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 ©2012 Intel Corporation.
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 SSE3 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