OpenMP: The *"Easy"* Path to Shared Memory Computing

Tim Mattson

Intel Corp.

timothy.g.mattson@intel.com

Disclaimer

READ THIS … it is very important

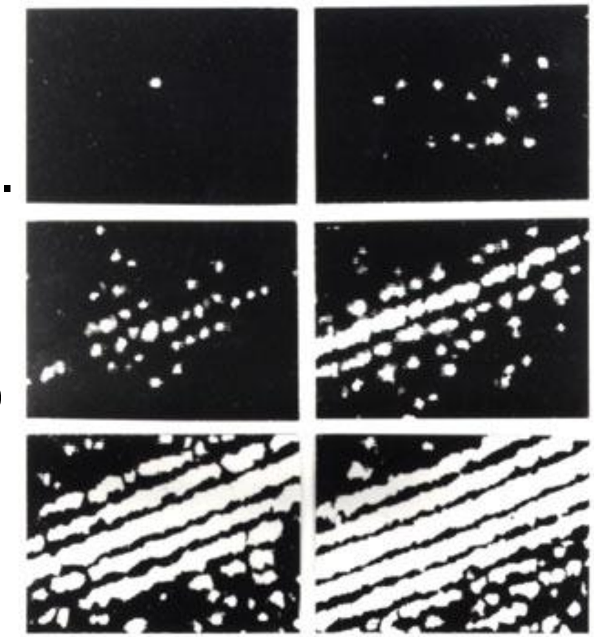
- The views expressed in this talk are those of the speaker and not his employer.

- This was a team effort, but if I say anything really stupid, it's my fault … don't blame my collaborators.

- A comment about performance data:
  - 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.

My scientific roots

- Quantum Mechanics (QM) changed my life.
  - Before QM… I was a chemistry major with a pre-med focus.
  - After QM… I dropped pre-med; determined to do whatever it took to understand quantum physics.



Interference patterns … 1 electron at a time passing through 2 slits

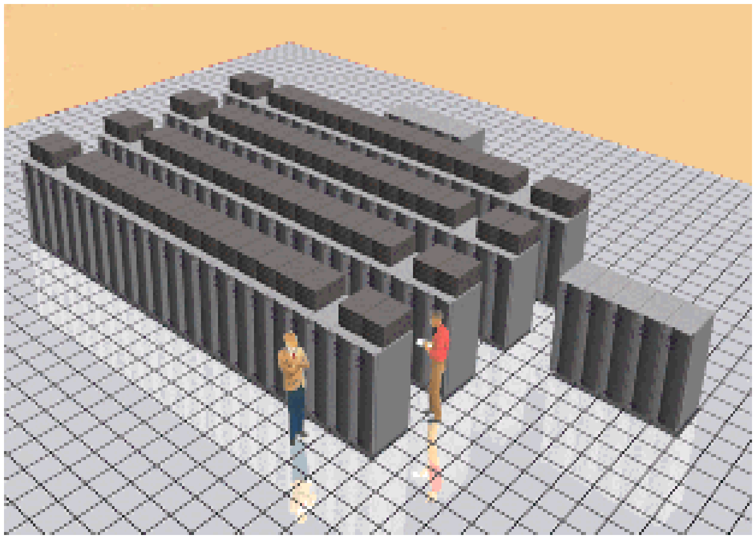
- I received a Ph.D. for work on quantum reactive scattering. To do this I had to …
  - Be a physicist to create useful but solvable model problems.
  - Be a mathematician to turn complex differential equations into solvable algebraic equations.
  - Be a computer scientist to map algorithms onto our primitive computers (VAX at 0.12 MFLOPS … compared to an iPhone today at 126 MFLOPS).

VAX 11/750, ~1980,

My career: The intersection of math, science and computer engineering

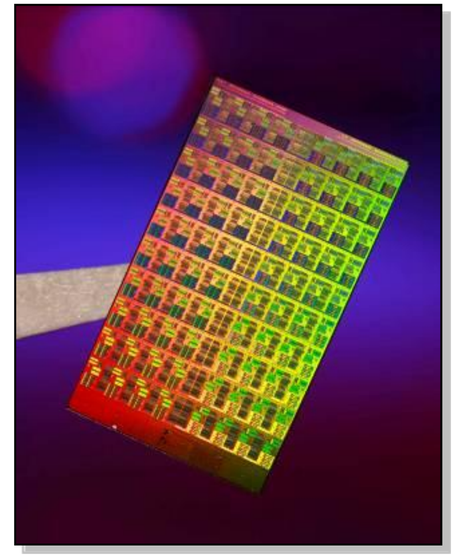
First TeraScale* computer: 1997

First TeraScale% chip: 2007



Intel's ASCI Option Red

10 years later



Intel's ASCI Red Supercomputer

9000 CPUs

one megawatt of electricity.

1600 square feet of floor space.

*Double Precision TFLOPS running MP-Linpack

Intel's 80 core teraScale Chip

1 CPU

97 watt

275 mm2

%Single Precision TFLOPS running stencil

A TeraFLOP in 1996: The ASCI TeraFLOP Supercomputer, Proceedings of the International Parallel Processing Symposium (1996), T.G. Mattson, D. Scott and S. Wheat.

Source: Intel

Professional goal: solve the many core challenge

- A harsh assessment …
  - We have turned to multi-core chips not because of the success of our parallel software but because of our failure to continually increase CPU frequency.

- Result: a fundamental and dangerous mismatch
  - Parallel hardware is ubiquitous … Parallel software is rare

- The Many Core challenge …
  - Parallel software must become as common as parallel hardware.

- Over the years I've worked on a number of parallel programming "languages".

Linda®

OpenMP™

OpenCL

STRAND88 MPI

3rd party names are the property of their owners.

Professional goal: solve the many core challenge

- A harsh assessment …
  - We have turned to multi-core chips <u>not</u> because of the success of our parallel software but because of <u>our failure</u> to continually increase CPU frequency.

- Result: a fundamental and dangerous mismatch
  - Parallel hardware is ubiquitous … Parallel software is rare

- The Ma
  - Paral … hardware.

Let's take a closer look at one of the most successful Parallel Programming Languages in use today ….

- Over the years I've worked on a number of parallel programming "languages".

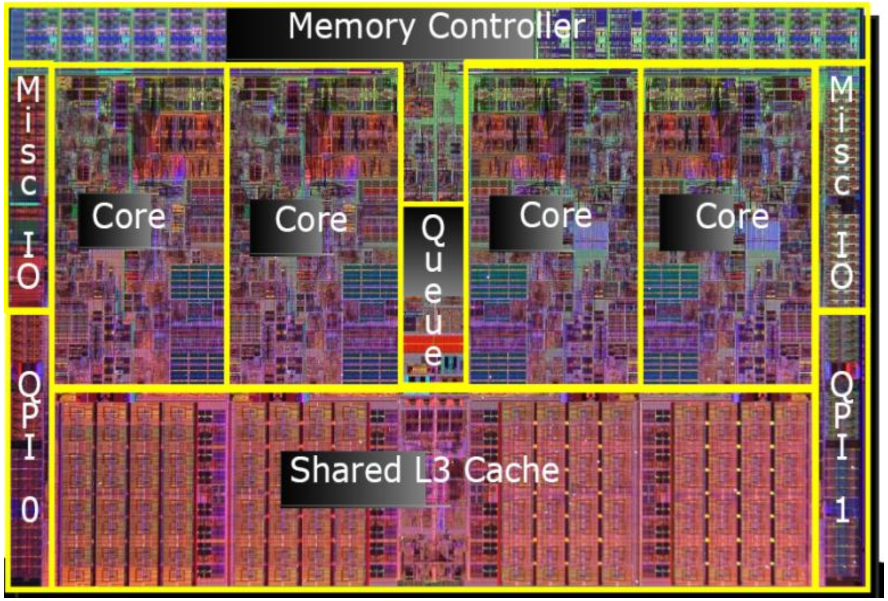
STRAND88

MP

3rd party names are the property of their owners.

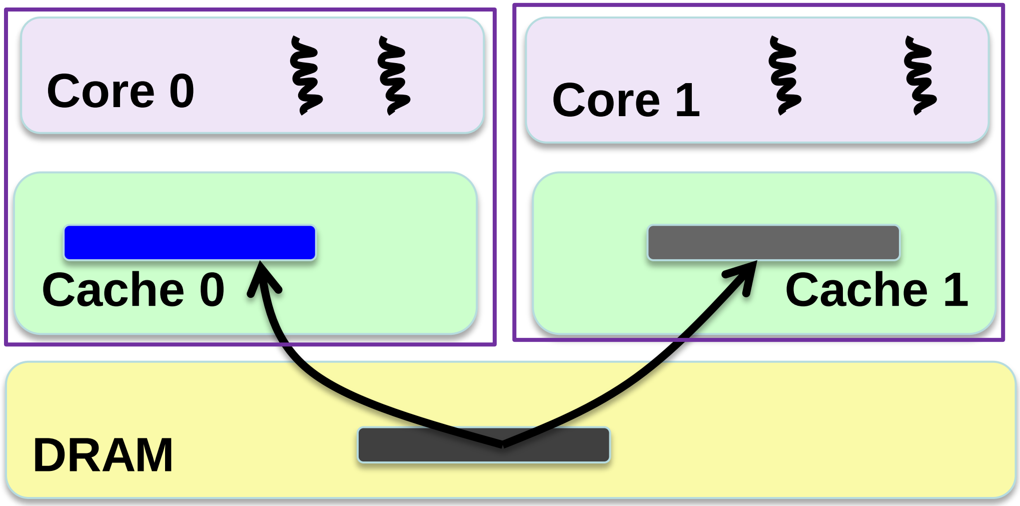
Assumptions

Intel™ Core™ i7 processor (Nehalem)



- You know about parallel architectures … multicore chips have made them very common.

- You know about threads and cache coherent shared address spaces



OpenMP* Overview:

C$OMP FLUSH

#pragma omp critical

C$OMP THREADPRIVATE(/ABC/)

CALL OMP_SET_NUM_THREADS(10)

C$OM

C$OM

C$O

ED

C

#p

OpenMP: An API for Writing Multithreaded Applications ... created in 1997

- A set of compiler directives and library routines for parallel application programmers
- Greatly simplifies writing multi-threaded (MT) programs in Fortran, C and C++
- Standardized years of SMP practice

C$OMP PARALLEL COPYIN(/blk/)

C$OMP DO lastprivate(XX)

Nthrds = OMP_GET_NUM_PROCS()

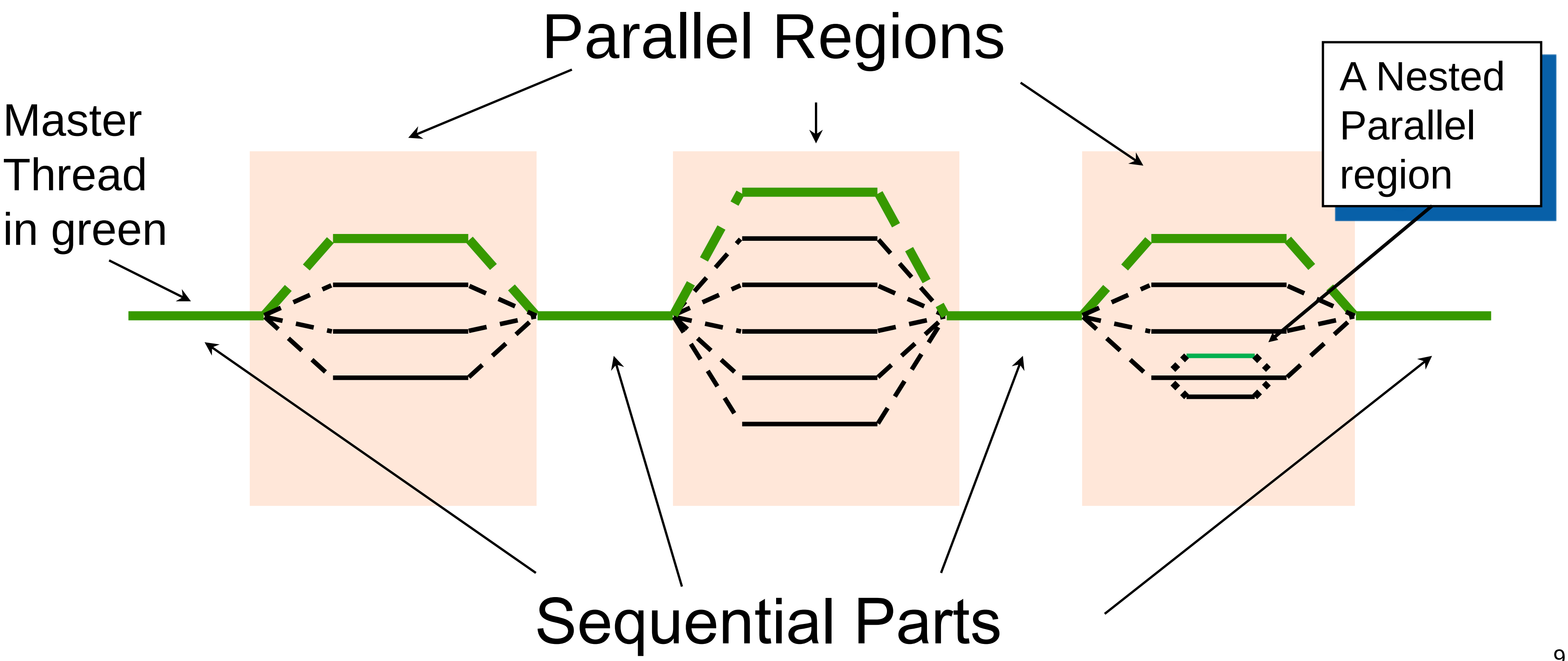
omp_set_lock(lck)

* The name "OpenMP" is the property of the OpenMP Architecture Review Board.

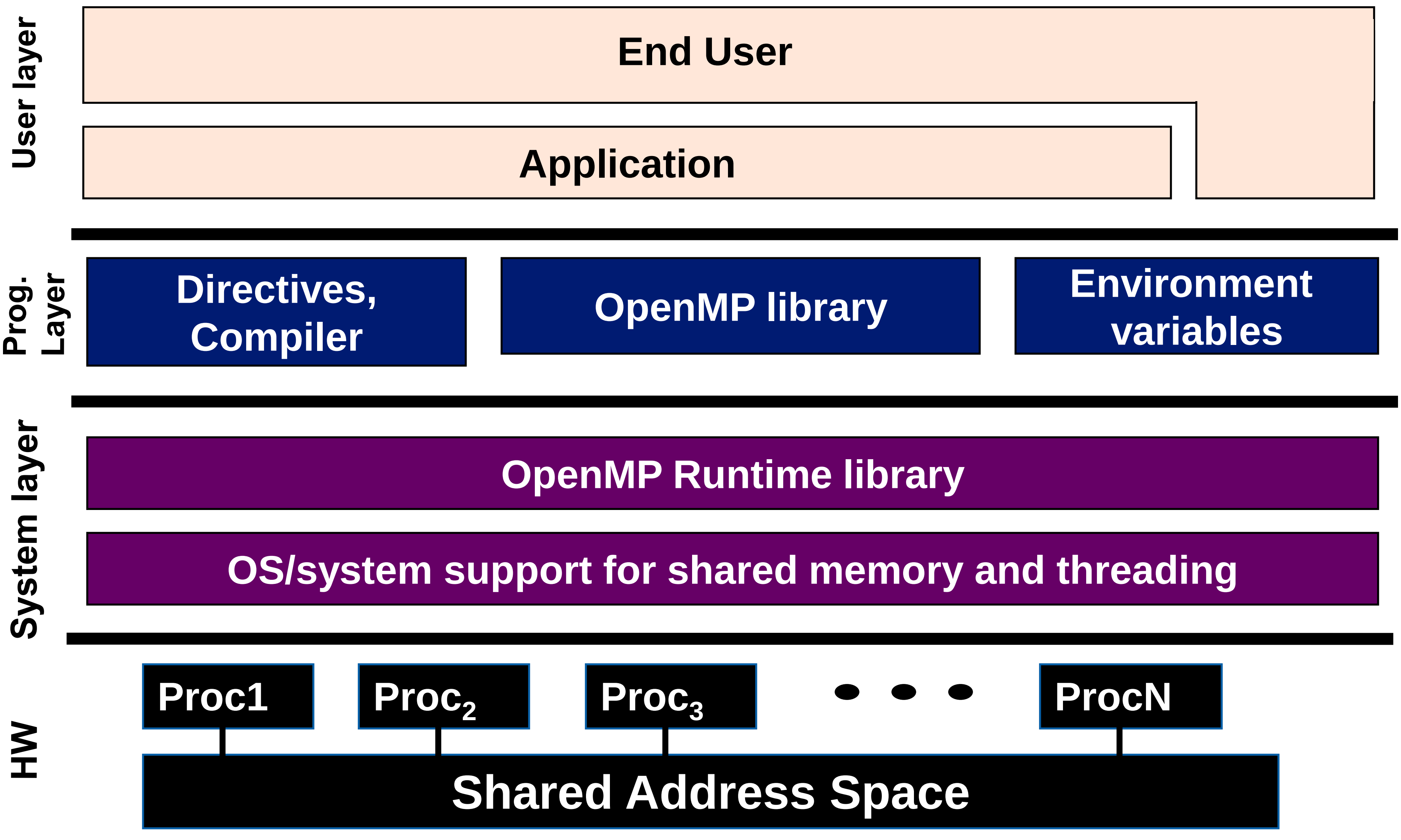
OpenMP Execution Model:

Fork-Join pattern:

- **Master thread** spawns a **team of threads** as needed.
- Parallelism added incrementally until performance goals are met: i.e. the sequential program evolves into a parallel program.



OpenMP Basic Defs: Solution Stack



Example: Hello world

- Write a multithreaded program where each thread prints "hello world".

```
void main()
{



    int ID = 0;
    printf(“ hello(%d) ”, ID);
    printf(“ world(%d) \n”, ID);

}
```

Example: Hello world Solution

- Tell the compiler to pack code into a function, fork the threads, and join when done …

```
#include "omp.h"
void main()
{

#pragma omp parallel
 {

    int ID = omp_get_thread_num();
    printf(" hello(%d) ", ID);
    printf(" world(%d) \n", ID);
  }
}
```

OpenMP include file

Parallel region with default number of threads

Runtime library function to return a thread ID.

End of the Parallel region

Sample Output:

hello(1) hello(0) world(1)

world(0)

hello (3) hello(2) world(3)

world(2)

OpenMP core syntax

- Most of the constructs in OpenMP are compiler directives.

  #pragma omp construct [clause [clause]…]

  – Example

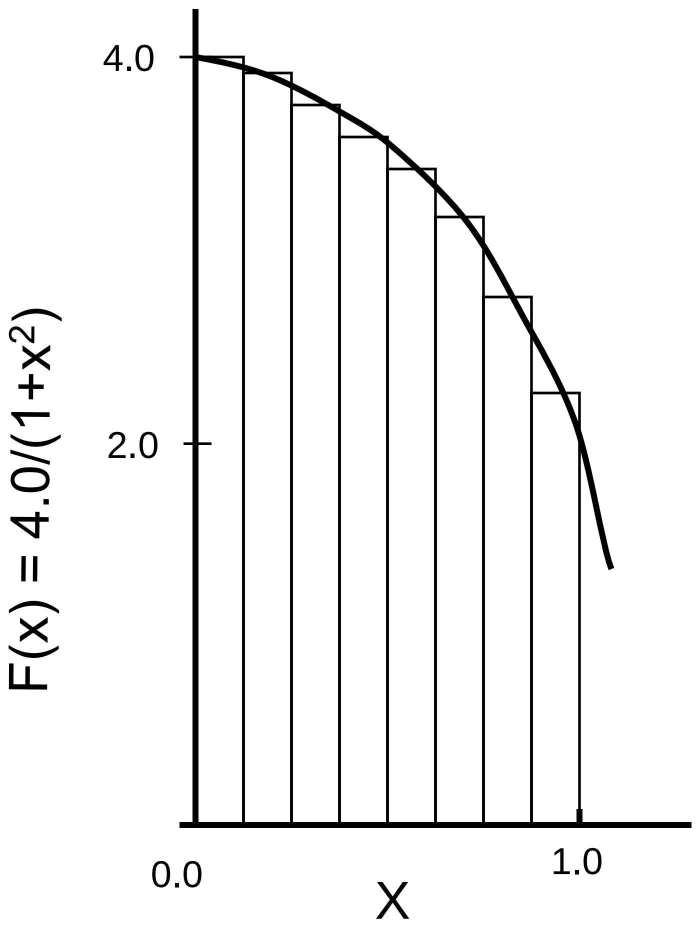
  #pragma omp parallel num_threads(4)

- Function prototypes and types in the file:

  #include <omp.h>

- Most OpenMP* constructs apply to a “structured block”.
  – Structured block: a block of one or more statements with one point of entry at the top and one point of exit at the bottom.
  – It’s OK to have an exit() within the structured block.

A simple running example: Numerical Integration



Mathematically, we know that:

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.

PI Program: Serial version

```
#define NUMSTEPS = 100000;
double step;
void main ()
{           int i;    double x, pi, sum = 0.0;

            step = 1.0/(double) NUMSTEPS;
            x = 0.5 * step;
            for (i=0;i<= NUMSTEPS; i++){
                    x+=step;
                    sum += 4.0/(1.0+x*x);
            }
            pi = step * sum;
}
```

A “simple” pi program

```
#include <omp.h>
static long num_steps = 100000;        double step;
Int main ()
{           double  pi;         step = 1.0/(double) num_steps;
#pragma omp parallel num_threads(4)
    {
            int i, id,nthrds;    double x, sum;
            id = omp_get_thread_num();
            nthrds = omp_get_num_threads();
            if (id == 0)   nthreads = nthrds;
             id = omp_get_thread_num();
            nthrds = omp_get_num_threads();
             for (i=id, sum=0.0;i< num_steps; i=i+nthreads){
                        x = (i+0.5)*step;
                        sum += 4.0/(1.0+x*x);
             }
           #pragma omp critical
                    pi += sum * step;
    }
}
```

This is a common trick in SPMD* programs to create a cyclic distribution of loop iterations

*SPMD = Single Program Multiple Data

Results*: pi program critical section

- Original Serial pi program with 100000000 steps ran in 1.83 seconds.

A "simple" pi program

```
#include <omp.h>
static long num_steps = 100000;        double  step;
Int main ()
{           double   pi;        step = 1.0/(double)  num_steps;
#pragma omp parallel num_threads(4)
    {
            int i, id,nthrds;    double  x, sum;
            id = omp_get_thread_num();
            nthrds  = omp_get_num_threads();
            if (id == 0)   nthreads  = nthrds;
             id = omp_get_thread_num();
             nthrds = omp_get_num_threads();
             for (i=id, sum=0.0;i< num_steps; i=i+nthreads){
                        x = (i+0.5)*step;
                        sum += 4.0/(1.0+x*x);
             }
            #pragma omp critical
                    pi += sum * step;
    }
}
```

This is a common trick in SPMD* programs to create a cyclic distribution of loop iterations

*SPMD = Single Program Multiple Data

16

threads	SPMD critical
1	1.87
2	1.00
3	0.68
4	0.53

*Intel compiler (icpc) with no optimization on Apple OS X 10.7.3 with a dual core (four HW thread) Intel® Core^{TM} i5 processor at 1.7 Ghz and 4 Gbyte DDR3 memory at 1.333 Ghz.

Loop worksharing Constructs
A motivating example

Sequential code

```
for(i=0;i<N;i++)   { a[i] = a[i] + b[i];}
```

OpenMP parallel region

```
#pragma omp parallel
{
        int id, i, Nthrds, istart, iend;
        id = omp_get_thread_num();
        Nthrds = omp_get_num_threads();
        istart = id * N / Nthrds;
        iend = (id+1) * N / Nthrds;
        if (id == Nthrds-1)iend = N;
        for(i=istart;i<iend;i++)   { a[i] = a[i] + b[i];}
}
```

OpenMP parallel region and a worksharing for construct

```
#pragma omp parallel
#pragma omp for
        for(i=0;i<N;i++)   { a[i] = a[i] + b[i];}
```

Reduction

- OpenMP reduction clause:

  reduction (op : list)

- Inside a parallel or a work-sharing construct:
  - A local copy of each list variable is made and initialized depending on the “op” (e.g. 0 for “+”).
  - Updates occur on the local copy.
  - Local copies are reduced into a single value and combined with the original global value.
- The variables in “list” must be shared in the enclosing parallel region.

```
double  ave=0.0, A[MAX];    int i;
#pragma omp parallel for reduction (+:ave)
for (i=0;i< MAX; i++) {
    ave + = A[i];
}
ave = ave/MAX;
```

Example: Pi with a loop and a reduction

```
#include <omp.h>
static long num_steps = 100000;  double step;

void main ()
{          int i;    double x, pi, sum = 0.0;
           step = 1.0/(double) num_steps;
#pragma omp parallel for private(x) reduction(+:sum)
           for (i=0;i< num_steps; i++){
                  x = (i+0.5)*step;
                  sum = sum + 4.0/(1.0+x*x);
           }
           pi = step * sum;
}
```

i private by default

Note: we created a parallel program without changing any executable code and by adding 2 simple lines of text!

Results*: pi with a loop and a reduction

- Original Serial pi program with 100000000 steps ran in 1.83 seconds.

Example: Pi with a loop and a re

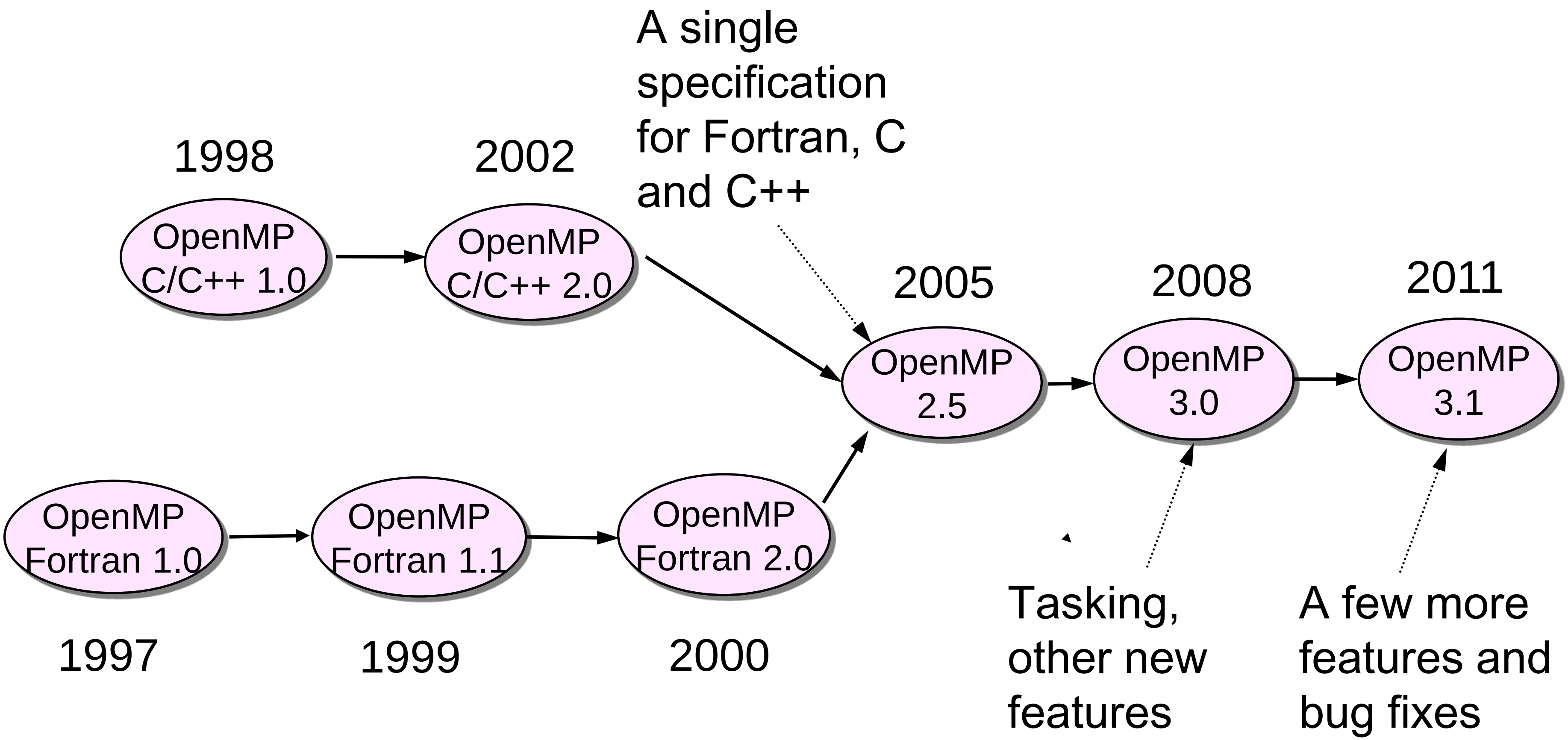
```
#include <omp.h>
static long num_steps = 100000;        double s

void main ()
{          int i;    double x, pi, sum = 0.0;
           step = 1.0/(double) num_steps;
#pragma omp parallel for private(x) reduction(+:sum)
           for (i=0;i< num_steps; i++){
                     x = (i+0.5)*step;
                     sum = sum + 4.0/(1.0+x*x);
           }
           pi = step * sum;
}
```

20

threads	SPMD critical	PI Loop
1	1.87	1.91
2	1.00	1.02
3	0.68	0.80
4	0.53	0.68

*Intel compiler (icpc) with no optimization on Apple OS X 10.7.3 with a dual core (four HW thread) Intel® Core^{TM} i5 processor at 1.7 Ghz and 4 Gbyte DDR3 memory at 1.333 Ghz.

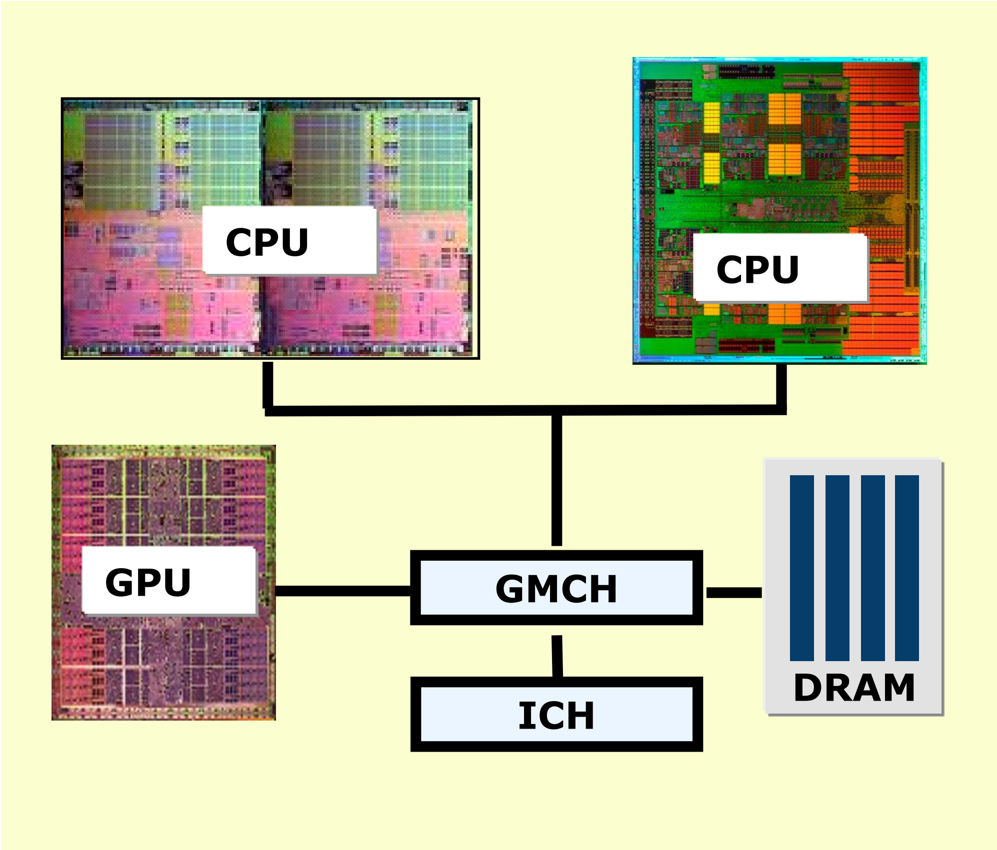
OpenMP Release History



What's next for OpenMP? Support for Heterogeneous systems

It's a Heterogeneous world

- A modern platform Includes:
  - One or more CPUs
  - One or more GPUs
  - DSP processors
  - … other?



OpenCL lets Programmers write a single portable program that uses ALL resources in the heterogeneous platform

GMCH = graphics memory control hub, ICH = Input/output control hub

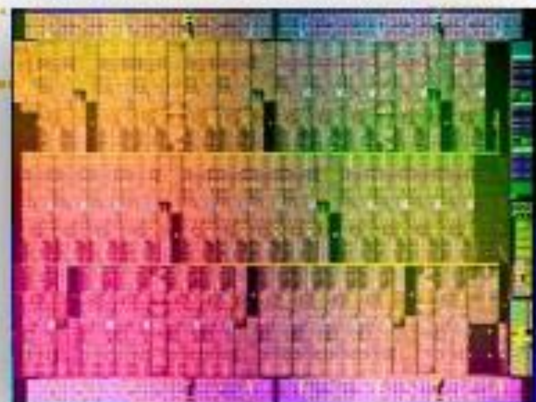
Overview – A Step Forward in Performance with Excellent Programmability

First product

Intel® Xeon PHI™ coprocessor … to be launched at SC12

Delivered Performance

Launching on 22nm with >50 cores to provide outstanding performance for HPC users



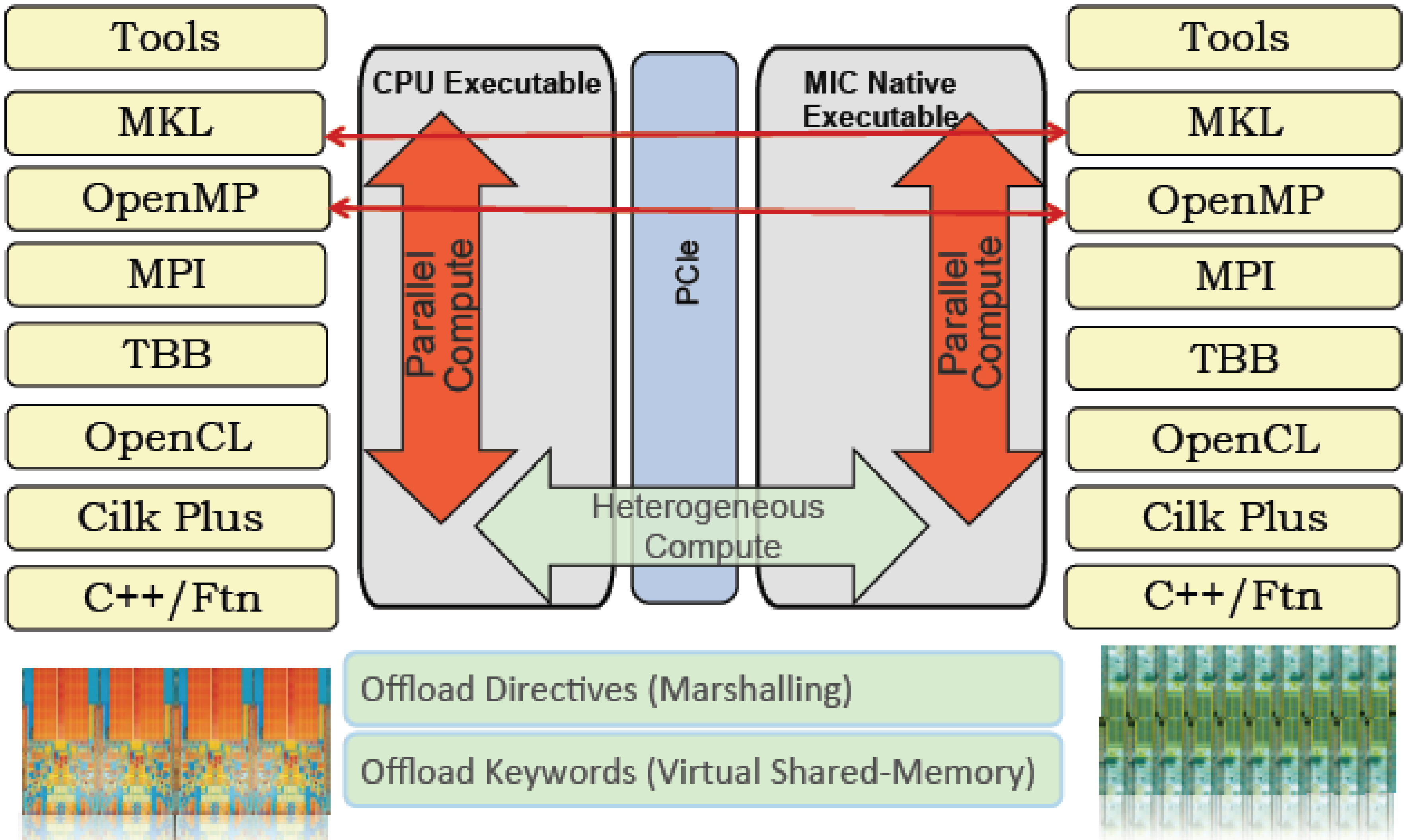
Performance Density

The compute density associated with specialty accelerators for parallel workloads

Programmability

The many benefits of broad Intel® processor programming models, techniques, and familiar x86 developer tools

Heterogeneous (Offload) Model



Parallel programming is the same on Intel® MIC and CPU

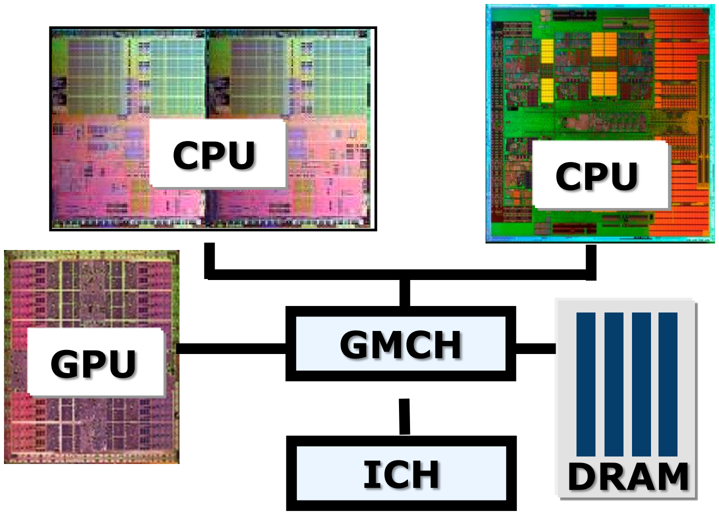
Example: Pi program … MIC Offload model

Intel has defined an offload API for manycore coprocessors

```
#include <omp.h>
static long num_steps = 100000;        double step;

void main ()
{            int i;      double x, pi, sum = 0.0;
             step = 1.0/(double) nsteps;
#pragma offload target (mic) in(nsteps, step) inout (sum, pi)
#pragma omp parallel for private(x) reduction(+:sum)
             for (i=0;i< num_steps; i++){
                         x = (i+0.5)*step;
                         sum = sum + 4.0/(1.0+x*x);
             }
             pi = step * sum;
}
```

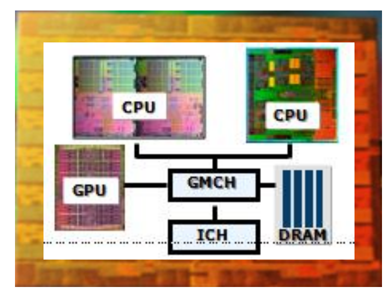
The OpenMP group is working a much more expansive set of directives for heterogeneous programming … which Intel compilers will support in early 2013. Attend the OpenMP BOF to learn more (Tues, 5:30 PM, room 355A)

Accelerators/coprocessors will go away … they are a temporary fad



- A modern platform includes:
  - CPU(s)
  - GPU(s)
  - DSP processors
  - … other?

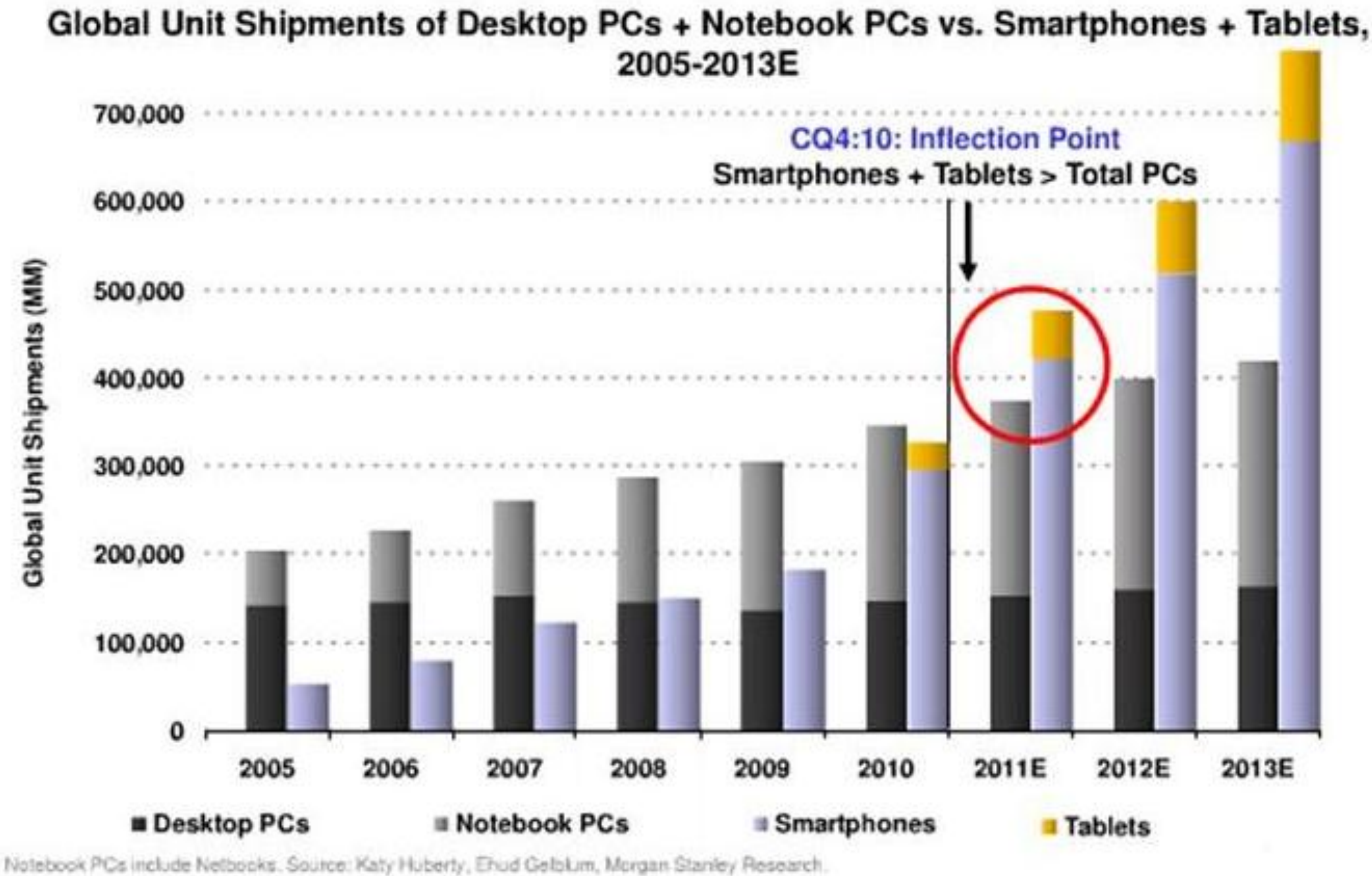
- And System on a Chip (SOC) trends are putting this all onto one chip



The future belongs to heterogeneous, many core SOC as the standard building block of computing

GMCH = graphics memory control hub, ICH = Input/output control hub SOC = system on a chip

Is the next great industry shake-up in progress?



Source: http://www.thephonelocker.com/articles/27249/device-diversity-is-your-new-normal-part-2/

Conclusion

- OpenMP is one of the simplest APIs available for programming shared memory machines.

- We provided enough of OpenMP to get you started, but there is much we didn't cover:
  - tasks
  - Additional work-share constructs
  - Detailed control over the data environment
  - …. And much more

- Heterogeneous computing is the latest development … and the new Intel® Xeon PHI™ coprocessor will be an interesting new player as a hybrid CPU-like many core device.