Introduction into Intel Xeon Phi Programming
LRZ, 28.4.2015

Dr. Volker Weinberg, weinberg@lrz.de
Intel Xeon Phi: Programming Models
MIC Programming Models

Xeon centric

Xeon hosted

offload

symmetric

MIC centric

MIC hosted

General purpose serial & parallel computing

Codes with highly parallel parts

Codes with balanced needs

Highly parallel codes

main()

foo()

MPI_*()

main()

foo()

MPI_*()

main()

foo()

MPI_*()

foo()

main()

foo()

MPI_*()

main()

foo()

MPI_*()
Advantages of the MIC Architecture

- Retains programmability and flexibility of standard x86 architecture.
- No need to learn a new complicated language like CUDA or OpenCL.
- Offers possibilities we always missed on GPUs: Login onto the system, watching and controlling processes via `top`, `kill` etc. like on a Linux host.
- Allows many different parallel programming models like OpenMP, MPI, Intel Cilk and Intel Threading Building Blocks.
- Offers standard math-libraries like Intel MKL.
- Supports whole Intel tool chain, e.g. Intel C/C++ and Fortran Compiler, Debugger & Intel VTune Amplifier.
Programming Modes

● Native Mode
  – Programs started on Xeon Phi.
  – Cross-compilation using –mmic.
  – User access to Xeon Phi necessary.
  – Necessary to support MPI ranks on Xeon Phi.

● Offload (Accelerator) Mode
  – Programs started on the host.
  – Intel Pragmas to offload code to Xeon Phi.
  – OpenMP possible, but no MPI ranks on Xeon Phi.
  – No user access to Xeon Phi needed.
  – No input data files on Xeon Phi possible.
  – Currently only mode at RZG, helios.
Offload Modes

- Host and MIC do not share physical or virtual memory in hardware.
- 2 Offload data transfer models are available:
  1. **Explicit copy**: Language Extensions for Offload (LEO)
     - Syntax: pragma/directive based
     - offload directive specifies variables that need to be copied between host and MIC
     - Example:
       - C: #pragma offload target(mic) in(data:length(size))
       - Fortran: !DIR$ offload target(mic) in(data:length(size))
  2. **Implicit Copy**: MYO
     - Syntax: keyword extension based
     - **shared variables need to be declared**, same variables can be used on the host and MIC, runtime automatically maintains coherence
     - Example:
       - C: _Cilk_shared double a; _Cilk_offload func(a);
       - Fortran: not supported
Programming Languages / Libraries

- **OpenMP**
  - Native execution on MIC (cross-compilation with \texttt{–mmic})
  - Execution on host, using offload pragmas / directives to offload code at runtime

- **MPI (and hybrid MPI & OpenMP)**
  - Co-processor only MPI programming model: native execution on MIC using 	exttt{mpiexec.hydra} on MIC.
  - Symmetric MPI programming model: MPI ranks on MICs and host CPUs.

- **MKL**
  - Native execution on MIC (compilation with \texttt{–mkl -mmic}).
  - Compiler assisted offload.
  - Automatic Offload (AO): automatically uses both host and MIC, transparent and automatic data transfer and execution management (compilation with \texttt{–mkl, mkl_mic_enable()} / \texttt{MKL\_MIC\_ENABLE=1}).
Distributed vs. Shared Memory

**Distributed Memory**

- **Same program** on each processor/machine (SPMD) or **Multiple programs** with consistent communication structure (MPMD)

- Program written in a sequential language
  - all variables process-local
  - no implicit knowledge of data on other processors

- Data exchange between processes:
  - send/receive messages via appropriate library
  - most tedious, but also the most flexible way of parallelization

- Parallel library discussed here:
  - Message Passing Interface, **MPI**

**Shared Memory**

- **Single Program** on single machine
  - UNIX Process splits off **threads**, mapped to CPUs for work distribution

- **Data**
  - may be process-global or thread-local
  - exchange of data not needed, or via suitable synchronization mechanisms

- **Programming models**
  - explicit threading (hard)
  - directive-based threading via **OpenMP** (easier)
  - automatic parallelization (very easy, but mostly not efficient)
MPI vs. OpenMP

- **MPI standard**
  - MPI forum released version 2.2 in September 2009
  - MPI version 3.0 in September 2012
  - unified document ("MPI1+2")

- **Base languages**
  - Fortran (77, 95)
  - C
  - C++ binding obsolescent → use C bindings

- **Resources:**
  - http://www.mpi-forum.org

- **OpenMP standard**
  - OpenMP 3.1 (July 2011) released by architecture review board (ARB)
    - feature update (tasking etc.)
  - OpenMP 4.0 (July 2013)
    - SIMD, affinity policies, **accelerator support**

- **Base languages**
  - Fortran (77, 95)
  - C, C++
    - (Java is not a base language)

- **Resources:**
  - http://www.openmp.org
  - http://www.compunity.org

---

**Portability:**
- of semantics (well-defined, "safe" interfaces)
- of performance (a difficult target)
Simple OpenMP program

```c
#include <omp.h>

int main() {
    int numth = 1;
    #pragma omp parallel
    {
        int myth = 0; /* private */
        #pragma omp single
        numth =omp_get_num_threads();
        /* block above: one statement */
        myth = omp_get_thread_num();

        printf("Hello from %i of %i\n",\n               myth,numth);
    } /* end parallel */
}

icc -openmp helloopenmp.c
```
Simple OpenMP Program

lu65fok@login12:~:/mickurs> export OMP_NUM_THREADS=10
lu65fok@login12:~:/mickurs> ./helloopenmp
Hello from 5 of 10
Hello from 2 of 10
Hello from 6 of 10
Hello from 0 of 10
Hello from 8 of 10
Hello from 3 of 10
Hello from 4 of 10
Hello from 9 of 10
Hello from 7 of 10
Hello from 1 of 10
/* C Example */
#include <stdio.h>
#include <mpi.h>
int main (int argc, char* argv[])
{
    int rank, size;

    MPI_Init (&argc, &argv);  /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);  /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size);  /* get number of processes */
    printf("Hello from %i of %i\n", rank, size);
    MPI_Finalize();
    return 0;
}

mpiicc hellompi.c
Simplest MPI Program

lu65fok@login12:~/> mpiicc hellompi.c -o hellompi

lu65fok@login12:~/> mpirun -n 10 ./hellompi
Hello from 5 of 10
Hello from 6 of 10
Hello from 7 of 10
Hello from 8 of 10
Hello from 9 of 10
Hello from 0 of 10
Hello from 1 of 10
Hello from 2 of 10
Hello from 3 of 10
Hello from 4 of 10
Intel Xeon Phi Programming Models: Native Mode
Intel compiler

- Initialisation integrated in our module system.
- Use `icpc / ifort --mmic` to compile for MIC

- On other systems you might need to run:

  ```
  lu65fok@login12:~> . $ICC_BASE/bin/compilervars.sh intel64
  ```

- This sets up important environment variables, e.g.
  - `MIC_LD_LIBRARY_PATH=\`
    `/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/compiler/lib/mic:\`
    `/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/mpirt/lib/mic:\`
    `/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/mkl/lib/mic:\`
    `/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/tbb/lib/mic:
  ```
GNU compiler: MIC support

● GNU Tools under /usr/linux-k1om-4.7/bin/x86_64-k1om-linux-*
  – addr2line, ar, as, c++filt, cpp, elfedit, g++, gcc, gcov, gprof, ld, ld.bfd, nm, objcopy, objdump, ranlib, readelf, size, strings, strip

● Compiler
  – Used to recompile the Linux kernel
  – To compile for MIC:
    lu65fok@i01r13c01:~/test> /usr/linux-k1om-4.7/bin/x86_64-k1om-linux-gcc hello.c -o hello

● GCC not recommended for Xeon Phi Programming!!
Useful Tools and Files on Coprocessor

- **top** - display Linux tasks
- **ps** - report a snapshot of the current processes.
- **kill** - send signals to processes, or list signals
- **ifconfig** - configure a network interface
- **traceroute** - print the route packets take to network host
- **mpiexec.hydra** – run Intel MPI natively
- **/proc/cpuinfo**
- **/proc/meminfo**
processor : 0
vendor_id : GenuineIntel
cpu family : 11
model : 1
model name : 0b/01
stepping : 3
cpu MHz : 1052.630
cache size : 512 KB
physical id : 0
siblings : 240
core id : 59
cpu cores : 60
apicid : 236
initial apicid : 236
fpu : yes
fpu_exception : yes
cpuid level : 4
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic mtrr mca pat fxsr ht
syscall nx lm rep_good nopl lahf_lm
bogomips : 2094.86
cflflush size : 64
cache_alignment : 64
address sizes : 40 bits physical, 48 bits virtual
power management:
```bash
[lu65fok@i01r13c01-mic0 proc]$ cat meminfo
MemTotal:   7882368 kB
MemFree:    7182704 kB
Buffers:    0 kB
Cached:     298824 kB
SwapCached: 0 kB
Active:     38660 kB
Inactive:   265544 kB
Active(anon): 38660 kB
Inactive(anon): 265544 kB
Active(file): 0 kB
Inactive(file): 0 kB
Unevictable: 0 kB
Mlocked:    0 kB
SwapTotal:  0 kB
SwapFree:   0 kB
Dirty:      0 kB
Writeback:  0 kB
...
Native Mode

● Compile on the Host (Login Node supermic):

```bash
lu65fok@login12:~$ . $ICC_BASE/bin/compilervars.sh intel64
lu65fok@login12:~$ icpc -mmic hello.c -o hello

lu65fok@login12:~$ . $IFORT_BASE/bin/compilervars.sh intel64
lu65fok@login12:~$ ifort -mmic hello.f90 -o hello
```

● Launch execution from the MIC:

```bash
lu65fok@login12:~$ scp hello i01r13c01-mic0:
hello                          100%  10KB 10.2KB/s 00:00
lu65fok@login12:~$ ssh i01r13c01-mic0
[lu65fok@i01r13c01-mic0 ~]$ ./hello
hello, world
lu65fok@i01r13c01-mic0 ~]$ Google Chrome
```

New: Home-Directories now also mounted on the MICs under /home/hpc/a2c06/lu23???
Native Mode: micnativeloadex

- Launch execution from the host:

  blu65fok@login12:~/test> ./hello
  -bash: ./hello: cannot execute Binary file

  lu65fok@i01r13c01:~/test> micnativeloadex ./hello
  hello, world

  lu65fok@i01r13c01:~/test> micnativeloadex ./hello -v
  hello, world
  Remote process returned: 0
  Exit reason: SHUTDOWN OK
MicInfo Utility Log

Created Thu Apr 17 17:22:27 2014

List of Available Devices

deviceld | domain | bus# | pciDev# | hardwareId
---------|--------|-----|--------|----------
  0 | 0 | 20 | 0 | 22508086
  1 | 0 | 8b | 0 | 22508086

-----------------------------------

28/04/2015 Intel Xeon Phi Programming
Number of Coprocessors

lu65fok@i01r13c01:~> micinfo | grep -i cores
Cores
    Total No of Active Cores : 60
Cores
    Total No of Active Cores : 60
lu65fok@i01r13c01:~>
lu65fok@login12:~/tests> cat hello.c
#include <unistd.h>
int main()
{
    printf("Hello world! I have \%ld logical cores.\n",
            sysconf(_SC_NPROCESSORS_ONLN));
}

lu65fok@i01r13c01:~/tests> ./hello-host
Hello world! I have 32 logical cores.

[lu65fok@i01r13c01-mic0 ~]$ ./hello-mic
Hello world! I have 240 logical cores.

lu65fok@i01r13c01:~/tests> micnativeloadex ./hello-mic
Hello world! I have 240 logical cores.

28/04/2015
Intel Xeon Phi Programming
Native Mode: micnativeloadex -l

lu65fok@i01r13c01:~/test> micnativeloadex hello -l

Dependency information for hello

Full path was resolved as
/home/hpc/pr28fa/lu65fok/test/hello

Binary was built for Intel(R) Xeon Phi(TM) Coprocessor
(codename: Knights Corner) architecture

SINK_LD_LIBRARY_PATH =

Dependencies Found:
  (none found)

Dependencies Not Found Locally (but may exist already on the coprocessor):
  libm.so.6
  libstdc++.so.6
  libgcc_s.so.1
  libc.so.6
  libdl.so.2

28/04/2015
Intel Xeon Phi Programming
micnativeloadex with Libraries

lu65fok@i01r13c01:/~tests> export SINK_LD_LIBRARY_PATH=./

lu65fok@i01r13c01:/~tests> micnativeloadex ./mylibtest -l

Dependency information for ./mylibtest

  Full path was resolved as
/home/hpc/pr28fa/lu65fok/tests=./mylibtest

  Binary was built for Intel(R) Xeon Phi(TM) Coprocessor
  (codename: Knights Corner) architecture

    SINK_LD_LIBRARY_PATH = ./

    Dependencies Found:

      /home/hpc/pr28fa/lu65fok/tests/mylib.so

    Dependencies Not Found Locally (but may exist already on the coprocessor):

      libimf.so
      libsvml.so
      libirng.so
      libm.so.6
      libstdc++.so.6
      libgcc_s.so.1
      libintel.so.5
      libc.so.6
      libdl.so.2
Lab: Native Mode
Intel Xeon Phi Programming Models: Explicit Offload Model
Intel Offload Directives

• Syntax:

  - C:
    
    #pragma offload target(mic) <clauses>
    <statement block>
  
  - Fortran:
    
    !DIR$ offload target(mic) <clauses>
    <statement>
    
    !DIR$ omp offload target(mic) <clauses>
    <OpenMP construct>
Intel Offload Directive

- **C:**
  - Pragma can be before any statement, including a compound statement or an OpenMP parallel pragma

- **Fortran:**
  - If OMP is specified: the next line, other than a comment, must be an OpenMP PARALLEL, PARALLEL SECTIONS, or PARALLEL DO directive.
  - If OMP is not specified, next line must:
    - An OpenMP* PARALLEL, PARALLEL SECTIONS, or PARALLEL DO directive
    - A CALL statement
    - An assignment statement where the right side only calls a function
Offloading a code block in Fortran:

!DIR$ offload begin target(MIC)
...
!DIR$ end offload

Code block can include any number of Fortran statements, including DO, CALL and any assignments, but not OpenMP directives.
Intel Offload

- Implements the following steps:

1. Memory allocation on the MIC
2. Data transfer from the host to the MIC
3. Execution on the MIC
4. Data transfer from the MIC to the host
5. Memory deallocation on MIC
#include <stdio.h>
int main (int argc, char* argv[]) {

#pragma offload target(mic)
{
    printf("MIC: Hello world from MIC.\n");
}

printf( "Host: Hello world from host.\n");
}
PROGRAM HelloWorld

!DIR$ offload begin target(MIC)
PRINT *,'MIC: Hello world from MIC'
!DIR$ end offload

PRINT *,'Host: Hello world from host'
END
lu65fok@login12:~/$ tests> icpc offload1.c -o offload1

lu65fok@login12:~/$ tests> ./offload1
offload error: cannot offload to MIC - device is not available

lu65fok@i01r13c01:~/$ tests> ./offload1
Host: Hello world from host.
MIC: Hello world from MIC.
Intel Offload: Hello World in Fortran

```bash
lu65fok@login12:~/tests> ifort offload1.f90 -o offload1

lu65fok@login12:~/tests> ./offload1
offload error: cannot offload to MIC - device is not available

lu65fok@i01r13c01:~/tests> ./offload1
Host: Hello world from host.
MIC: Hello world from MIC.
```
#include <stdio.h>
#include <unistd.h>

int main (int argc, char* argv[]) {
    char hostname[100];
    gethostname(hostname,sizeof(hostname));

    #pragma offload target(mic)
    {
        char michostname[100];
        gethostname(michostname, sizeof(michostname));
        printf("MIC: Hello world from MIC. I am %s and I have %ld logical cores. I was called from host: %s \n", michostname, sysconf(_SC_NPROCESSORS_ONLN), hostname);
    }
}
lu65fok@login12:~/tests> icpc offload.c -o offload

lu65fok@i01r13c01:~/tests> ./offload
Host: Hello world from host. I am i01r13c01 and I have 32 logical cores.
MIC: Hello world from MIC. I am i01r13c01-mic0 and I have 240 logical cores. I was called from host: i01r13c01
lu65fok@login12:~/tests> icpc -offload=optional offload.c -o offload

lu65fok@login12:~/tests> ./offload
MIC: Hello world from MIC. I am login12 and I have 16 logical cores. I was called from host: login12
Host: Hello world from host. I am login12 and I have 16 logical cores.

lu65fok@login12:~/tests> icpc -offload=mandatory offload.c -o offload
lu65fok@login12:~/tests> ./offload
offload error: cannot offload to MIC - device is not available
lu65fok@login12:~/tests> icpc -offload=none offload.c -o offload
offload.c(13): warning #161: unrecognized #pragma
 #pragma offload target(mic)
 ^

lu65fok@login12:~/tests>

lu65fok@i01r13c01:~/tests> ./offload
MIC: Hello world from MIC. I am i01r13c01 and I have 32 logical cores.
I was called from host: i01r13c01
Host: Hello world from host. I am i01r13c01 and I have 32 logical cores.
#include <stdio.h>
#include <stdlib.h>

int main(){

#pragma offload target (mic)
{
    system("command");
}
}
Intel Offload: system("set")

lu65fok@i01r13c01:~/tests> ./system
BASH=/bin/sh
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_EXECUTION_STRING=set
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSION='4.2.10(1)-release'
COI_LOG_PORT=65535
COI_SCIF_SOURCE_NODE=0
DIRSTACK=()
ENV_PREFIX=MIC
EUID=400
GROUPS=()
HOSTNAME=i01r13c01-mic0
HOSTTYPE=k1om
IFS=''

LIBRARY_PATH=/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/tbb/lib/mic:/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/tbb/lib/mic:/lrz/sys/intel/compiler140_144/composer_xe_2013_sp1.2.144/tbb/lib/mic:
MACHTYPE=k1om-mpss-linux-gnu
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/bin:/bin
POSIXLY_CORRECT=y
PPID=37141
PS4='+ ' PWD=/var/volatile/tmp/coi_procs/1/37141
SHELL=/bin/false
SHELLOPTS=braceexpand:hashall:interactive-comments:posix
SHLVL=1
TERM=dumb
UID=400
_=sh
Offload: Using several MIC Coprocessors

- To query the number of coprocessors:
  
  ```c
  int nmics = __Offload_number_of_devices()
  ```

- To specify which coprocessor n< nmics should do the computation:
  
  ```c
  #pragma offload target(mic:n)
  ```

- If (n > nmics) then coprocessor (n % nmics) is used

- **Important for:**
  - Asynchronous offloads
  - Coprocessor-Persistent data
Offloading OpenMP Computations

- C/C++ & OpenMP:
  
  ```c
  #pragma offload target(mic)
  #pragma omp parallel for
  for (int i=0;i<n;i++) {
    a[i]=c*b[i]+d;
  }
  ```

- Fortran & OpenMP
  
  ```fortran
  !$DIR$ offload target(mic)
  !$OMP PARALLEL DO
  do i = 1, n
    a(i) = c*b(i) + d
  end do
  !$omp END PARALLEL DO
  ```
Functions and variables on the MIC

- **C:**
  - `__attribute__((target(mic)))` variables / function
  - `__declspec(target(mic))` variables / function
  - `#pragma offload_attribute(push, target(mic))` … multiple lines with variables / functions
  - `#pragma offload_attribute(pop)`

- **Fortran:**
  - `!DIR$ attributes offload:mic::` variables / function
Functions and variables on the MIC

```c
#pragma offload_attribute(push,target(mic))
const int n=100;
int a[n], b[n], c, d;
void myfunction(int* a, int*b, int c, int d){
    for (int i=0;i<n;i++) {
        a[i]=c*b[i]+d;
    }
}
#pragma offload_attribute(pop)

int main (int argc, char* argv[]) {
    #pragma offload target(mic)
    {
        myfunction(a,b,c,d);
    }
}```
# Intel Offload Clauses

<table>
<thead>
<tr>
<th>Clauses</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple coprocessors</td>
<td><code>target(mic[:unit])</code></td>
<td>Select specific coprocessors</td>
</tr>
<tr>
<td>Conditional offload</td>
<td><code>if (condition) / mandatory</code></td>
<td>Select coprocessor or host compute</td>
</tr>
<tr>
<td>Inputs</td>
<td><code>in(var-list modifiers_{opt})</code></td>
<td>Copy from host to coprocessor</td>
</tr>
<tr>
<td>Outputs</td>
<td><code>out(var-list modifiers_{opt})</code></td>
<td>Copy from coprocessor to host</td>
</tr>
<tr>
<td>Inputs &amp; outputs</td>
<td><code>inout(var-list modifiers_{opt})</code></td>
<td>Copy host to coprocessor and back when offload completes</td>
</tr>
<tr>
<td>Non-copied data</td>
<td><code>nocopy(var-list modifiers_{opt})</code></td>
<td>Data is local to target</td>
</tr>
<tr>
<td>Async. Offload</td>
<td><code>signal(signal-slot)</code></td>
<td>Trigger asynchronous Offload</td>
</tr>
<tr>
<td>Async. Offload</td>
<td><code>wait(signal-slot)</code></td>
<td>Wait for completion</td>
</tr>
</tbody>
</table>
## Intel Offload Modifier Options

<table>
<thead>
<tr>
<th>Modifiers</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify copy length</td>
<td><code>length(N)</code></td>
<td>Copy N elements of pointer’s type</td>
</tr>
<tr>
<td>Coprocessor memory allocation</td>
<td><code>alloc_if (bool)</code></td>
<td>Allocate coprocessor space on this offload (default: TRUE)</td>
</tr>
<tr>
<td>Coprocessor memory release</td>
<td><code>free_if (bool)</code></td>
<td>Free coprocessor space at the end of this offload (default: TRUE)</td>
</tr>
<tr>
<td>Array partial allocation &amp; variable relocation</td>
<td><code>alloc (array-slice) in ( var-expr )</code></td>
<td>Enables partial array allocation and data copy into other vars &amp; ranges</td>
</tr>
</tbody>
</table>
Intel Offload: Data movement

- #pragma offload target(mic) in(in1,in2,…)
  out(out1,out2,…) inout(inout1,inout2,…)

- **At Offload start:**
  - Allocate Memory Space on MIC for all variables
  - Transfer in/inout variables from Host to MIC

- **At Offload end:**
  - Transfer out/inout variables from MIC to Host
  - Deallocate Memory Space on MIC for all variables
Intel Offload: Data movement

- data = (double*)malloc(n*sizeof(double));
- #pragma offload target(mic) in(data:length(n))

- Copies n doubles to the coprocessor, not n * sizeof(double) Bytes
- ditto for out() and inout()
Allocation of Partial Arrays in C

- `int n=1000`
- `data = (double*)malloc(n*sizeof(double));`
- `#pragma offload target(mic) in(data[100:200] : alloc(data[300:400]))`

- **Host:**
  - 1000 doubles allocated
  - First element has index 0
  - Last element has index 999

- **MIC:**
  - 400 doubles are allocated
  - First element has index 300
  - Last element has index 699
  - 200 elements in the range `data[100], …, data[299]` are copied to the MIC
Allocation of Partial Arrays in Fortran

- integer :: n=1000
- double precision, allocatable :: data(:)
- allocate(data(n) )
- !C: #pragma offload target(mic) in(data[100:200] : alloc(data[300:400])
- !DIR$ offload target(mic) in(data(100:299) : alloc(data(300:699))
- **Host:**
  - 1000 doubles allocated
  - First element has index 0
  - Last element has index 999
- **MIC:**
  - 400 doubles are allocated
  - First element has index 300
  - Last element has index 699
  - 200 elements in the range data[100], …, data[299] are copied to the MIC
#pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
{
    #pragma omp parallel for
    for( i = 0; i < n; i++ ) {
        for( k = 0; k < n; k++ ) {
            #pragma vector aligned
            #pragma ivdep
            for( j = 0; j < n; j++ ) {
                //c[i][j] = c[i][j] + a[i][k]*b[k][j];
                c[i*n+j] = c[i*n+j] + a[i*n+k]*b[k*n+j];
            }
        }
    }
}
Vectorisation Diagnostics

lu65fok@login12:~/tests> icc -vec-report2 -openmp offloadmul.c -ooffloadmul
offloadmul.c(35): (col. 5) remark: LOOP WAS VECTORIZED
offloadmul.c(32): (col. 3) remark: loop was not vectorized: not inner loop
offloadmul.c(57): (col. 2) remark: LOOP WAS VECTORIZED
offloadmul.c(54): (col. 7) remark: loop was not vectorized: not inner loop
offloadmul.c(53): (col. 5) remark: loop was not vectorized: not inner loop
offloadmul.c(8): (col. 9) remark: loop was not vectorized: existence of vector dependence
offloadmul.c(7): (col. 5) remark: loop was not vectorized: not inner loop
offloadmul.c(57): (col. 2) remark: *MIC* LOOP WAS VECTORIZED
offloadmul.c(54): (col. 7) remark: *MIC* loop was not vectorized: not inner loop
offloadmul.c(53): (col. 5) remark: *MIC* loop was not vectorized: not inner loop

__attribute__((target(mic))) void mxm( int n, double * restrict a, double * restrict b, double * restrict c ){

    int i,j,k;
    for( i = 0; i < n; i++ ) {
        ...
    }

}

main(){
...
    #pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
    {
        mxm(n,a,b,c);
    }

}
Offload Diagnostics

u65fok@i01r13c06:~/tests> export OFFLOAD_REPORT=2

lu65fok@i01r13c06:~/tests> ./offloadmul

[Offload] [MIC 0] [File] offloadmul.c
[Offload] [MIC 0] [Line] 50
[Offload] [MIC 0] [Tag] Tag 0
[Offload] [HOST] [Tag 0] [CPU Time] 51.927456(seconds)
[Offload] [MIC 0] [Tag 0] [CPU->MIC Data] 24000016 (bytes)
[Offload] [MIC 0] [Tag 0] [MIC Time] 50.835065(seconds)
[Offload] [MIC 0] [Tag 0] [MIC->CPU Data] 8000016 (bytes)
Offload Diagnostics

lu65fok@i01r13c06:~/tests> export H_TRACE=1

lu65fok@i01r13c06:~/tests> ./offloadmul
HOST: Offload function
__offload_entry_offloadmul_c_50mainicc638762473Jnx4JU, is_empty=0, #varDescs=7, #waits=0, signal=None
HOST: Total pointer data sent to target: [24000000] bytes
HOST: Total copyin data sent to target: [16] bytes
HOST: Total pointer data received from target: [8000000] bytes
MIC0: Total copyin data received from host: [16] bytes
MIC0: Total copyout data sent to host: [16] bytes
HOST: Total copyout data received from target: [16] bytes
lu65fok@i01r13c06:~/tests>
Offload Diagnostics

lu65fok@i01r13c06:~/tests> export H_TIME=1

lu65fok@i01r13c06:~/tests> ./offloadmul

[Offload] [MIC 0] [File]   offloadmul.c
[Offload] [MIC 0] [Line]   50
[Offload] [MIC 0] [Tag]    Tag 0
[Offload] [HOST]  [Tag 0] [CPU Time]   51.920016(seconds)
[Offload] [MIC 0] [Tag 0] [MIC Time]   50.831497(seconds)

**********************************************************************************

             timer data   (sec)
**********************************************************************************

lu65fok@i01r13c06:~/tests>
Environment Variables

- Host environment variables are automatically forwarded to the coprocessor when offload mode is used.
- To avoid names collisions:
  - Set MIC_ENVIRONMENT_PREFIX=MIC on the host
  - Then only names with prefix MIC_ are forwarded to the coprocessor with prefix stripped
  - Exception: MIC_LD_LIBRARY_PATH is never passed to the coprocessor.
  - Value of LD_LIBRARY_PATH cannot be changed via forwarding of environment variables.
Environment Variables on the MIC

#include <stdio.h>
#include <stdlib.h>

int main()
{
    #pragma offload target (mic)
    {
        char* varmic = getenv("VAR");
        if (varmic) {
            printf("VAR=%s on MIC.\n", varmic);
        } else {
            printf("VAR is not defined on MIC.\n");
        }
    }
    char* varhost = getenv("VAR");
    if (varhost) {
        printf("VAR=%s on host.\n", varhost);
    } else {
        printf("VAR is not defined on host.\n");
    }
}
Environment Variables on the MIC

lu65fok@i01r13c01:~/tests> ./env
VAR is not defined on host.
VAR is not defined on MIC.
lu65fok@i01r13c01:~/tests> export VAR=299792458
lu65fok@i01r13c01:~/tests> ./env
VAR=299792458 on host.
VAR=299792458 on MIC.
lu65fok@i01r13c01:~/tests> export MIC_ENV_PREFIX=MIC
lu65fok@i01r13c01:~/tests> ./env
VAR=299792458 on host.
VAR is not defined on MIC.
lu65fok@i01r13c01:~/tests> export MIC_VAR=3.141592653
lu65fok@i01r13c01:~/tests> ./env
VAR=299792458 on host.
VAR=3.141592653 on MIC.
The Preprocessor Macro __MIC__

- The macro __MIC__ is only defined in code version for MIC, not in the fallback version for the host.
- Allows to check where the code is running.
- Allows to write multiversioned code.
- __MIC__ also defined in native mode.
The Preprocessor Macro __MIC__

```
#pragma offload target(mic)
{
    #ifdef __MIC__
        printf("Hello from MIC (offload succeeded).\n");
    #else
        printf("Hello from host (offload to MIC failed!).\n");
    #endif
}
```

```
lut65fok@login12:~/tests> icpc -offload=optional offload-mic.c
lut65fok@login12:~/tests> ./a.out
Hello from host (offload to MIC failed!).
lut65fok@i01r13c06:~/tests> ./a.out
Hello from MIC (offload succeeded).
```
Data Traffic without Computation

• 2 possibilities:
  
  – Blank body of `#pragma offload`, i.e.
    
    ```c
    #pragma offload target(mic) in (data: length(n))
    {}  
    ```

  – Use a special pragma `offload_transfer`, i.e.
    
    ```c
    #pragma offload_transfer target(mic) in(data: length(n))
    ```
Proxy Console I/O

- stderr and stdout on MIC are buffered and forwarded (proxied) to the host console.
- Forwarding is done by the coi_daemon running on the MIC.
- Output buffer should be flushed with `fflush(0)` of the `stdio`-Library.
- Proxy console input not supported.
- Proxy I/O is enabled by default.
- Can be switched off using `MIC_PROXY_IO=0`. 

28/04/2015
Intel Xeon Phi Programming
#include <stdio.h>
#include <unistd.h>

__attribute__((target(mic))) extern struct _IO_FILE *stderr;

int main (int argc, char* argv[]){
  char hostname[100];
  gethostname(hostname,sizeof(hostname));

  #pragma offload target(mic) {
    char michostname[100];
    gethostname(michostname, sizeof(michostname));
    printf("MIC stdout: Hello world from MIC. I am %s and I have %ld logical cores. I was called from host: %s \n", michostname, sysconf(_SC_NPROCESSORS_ONLN), hostname);
    fprintf(stderr,"MIC stderr: Hello world from MIC. I am %s and I have %ld logical cores. I was called from host: %s \n", michostname, sysconf(_SC_NPROCESSORS_ONLN), hostname);
    fflush(0);
  }

  printf("Host stdout: Hello world from host. I am %s and I have %ld logical cores.\n", hostname, sysconf(_SC_NPROCESSORS_ONLN));
  fprintf(stderr,"Host stderr: Hello world from host. I am %s and I have %ld logical cores.\n", hostname, sysconf(_SC_NPROCESSORS_ONLN));
lu65fok@i01r13c01:~/tests> ./proxyio 1>proxyio.out 2>proxyio.err

lu65fok@i01r13c01:~/tests> cat proxyio.out
MIC stdout: Hello world from MIC. I am i01r13c01-mic0 and I have 240 logical cores.
I was called from host: i01r13c01
Host stdout: Hello world from host. I am i01r13c01 and I have 32 logical cores.

lu65fok@i01r13c01:~/tests> cat proxyio.err
MIC stderr: Hello world from MIC. I am i01r13c01-mic0 and I have 240 logical cores. I
was called from host: i01r13c01
Host stderr: Hello world from host. I am i01r13c01 and I have 32 logical cores.
lu65fok@i01r13c01:~/tests>
Proxy Console I/O

```
lu65fok@i01r13c01:~/tests> export MIC_PROXY_IO=0

lu65fok@i01r13c01:~/tests> ./proxyio 1>proxyio.out 2>proxyio.err

lu65fok@i01r13c01:~/tests> cat proxyio.out
Host stdout: Hello world from host. I am i01r13c01 and I have 32 logical cores.

lu65fok@i01r13c01:~/tests> cat proxyio.err
Host stderr: Hello world from host. I am i01r13c01 and I have 32 logical cores.
```

28/04/2015
Intel Xeon Phi Programming
Asynchronous Offload

Asynchronous Data Transfer helps to:

- Overlap computations on host and MIC(s).
- Work can be distributed to multiple coprocessors.
- Data transfer time can be masked.
Asynchronous Offload

- To allow asynchronous data transfer, the specifies `signal()` and `wait()` can be used, i.e.

```c
#pragma offload_transfer target(mic:0) in(data : length(n))
signal(data)

// work on other data concurrent to data transfer …
#pragma offload target(mic:0) wait(data) \ nocopy(data : length(N)) out(result : length(N))
{
    ....
    result[i]=data[i] + ...;
}
```

Any pointer type variable can serve as a signal!

Device number must be specified!
Asynchronous Offload

- Alternative to the wait() clause, a new pragma can be used:
  
  ```
  #pragma offload_wait target(mic:0) wait(data)
  ```

- Useful if no other offload or data transfer is necessary at the synchronisation point.
char* offload0;
char* offload1;
#pragma offload target(mic:0) signal(offload0) \
   in(data0 : length(N)) out(result0 : length(N))
{
   Calculate(data0, result0);
}
#pragma offload target(mic:1) signal(offload1) \
   in(data1 : length(N)) out(result1 : length(N))
{
   Calculate(data1, result1);
}
#pragma offload_wait target(mic:0) wait(offload0) 
#pragma offload_wait target(mic:1) wait(offload1)
Explicit Worksharing

```c
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        {
            //section running on the coprocessor
            #pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n))
            {
                mxm(n,a,b,c);
            }
        }
        #pragma omp section
        {
            //section running on the host
            mxm(n,d,e,f);
        }
    }
}
```
# Persistent Data

- `#define ALLOC alloc_if(1)`
- `#define FREE free_if(1)`
- `#define RETAIN free_if(0)`
- `#define REUSE alloc_if(0)`

- To allocate data and keep it for the next offload:
  `#pragma offload target(mic) in (p:length(l) ALLOC RETAIN)`

- To reuse the data and still keep it on the coprocessor:
  `#pragma offload target(mic) in (p:length(l) REUSE RETAIN)`

- To reuse the data again and free the memory. (FREE is the default, and does not need to be explicitly specified):
  `#pragma offload target(mic) in (p:length(l) REUSE FREE)`
Further control on compiler flags

- Check the flags passed to the offload compiler:
  -watch=mic-cmd

- Flags:
  - -offload-option,mic,compiler,"-O3 -ip -vec-report6"
  - -offload-option,mic,link,"-L/my/path/lib -Lmymiclib"

```
lu65fok@login12:~/tests> icpc -watch=mic-cmd -offload-option,mic,compiler,"-O3 -ip -vec-report6" -offload-option,mic,link,"-L/my/path/lib -Lmymiclib" -offload-option,mic,link,"-L/my/path/lib -Lmymiclib" offload-mic.c

MIC command line:
icpc offload-mic.c -O3 -ip -vec-report6
```
Lab: Offload Mode
Intel Xeon Phi Programming Models: Intel Cilk Plus
Cilk Plus

- http://www.cilkplus.org/
- “The easiest, quickest way to harness the power of both multicore and vector processing.”
- Extension to the C and C++ languages to support data and task parallelism.
- Only 3 new keywords to implement task parallelism.
- Serial semantics make understanding and debugging the parallel program easier.
- Array Notations provide a natural way to express data parallelism.
The Three Cilk Keywords

- Intel Cilk Plus adds three keywords to C and C++ to allow developers to express opportunities for
  - `cilk_for` - Parallelize for loops
  - `cilk_spawn` - Specifies that a function can execute in parallel with the remainder of the calling function
  - `cilk_sync` - Specifies that all spawned calls in a function must complete before execution continues
Intel Cilk Plus C/C++ Extensions for Array Notations

- The C/C++ language extensions for array notations are Intel-specific language extensions that are part of the Intel® Cilk™ Plus feature.

- **Major benefits:**
  - Allows you to use array notation to program vector operations in a familiar language
  - Achieves predictable performance based on mapping parallel constructs to the underlying SIMD hardware
  - Enables compiler vectorization with less reliance on alias and dependence analysis

- **Tutorial:** http://www.cilkplus.org/tutorial-array-notation
Intel Cilk Plus C/C++ Extensions for Array Notations

• The array notations work at all optimization levels. Specifying -O0 compiler options serializes the array operations into sequential loops and helps to debug applications using array notations.

• \( a[\text{lb:}\text{len:}\text{str}] \) is an array section with len elements: \( a[\text{lb}], a[\text{lb} + \text{str}], a[\text{lb} + 2\times\text{str}], \ldots, a[\text{lb} + (\text{len}-1)\times\text{str}] \).

• Examples:
  - \( A[:] += B[:]; \)
  - \( A[0:16] += B[32:16]; \)
  - \( A[0:16:2] += B[32:16:4]; \)

Number of elements, not upper bound!
Intel Xeon Phi Programming Models: MYO
Virtual Shared Classes

- Offload Model only allows offloading of bitwise-copyable data.
- Sharing complicated structures with pointers or C++ classes is only possible via MYO.
class _Cilk_shared Person {
    public:
        int id;
        char name[10];
    Person() { id=0; name[0]=\0; }
    void Set(_Cilk_shared const char* name0, const int id0) {
        id = i0;
        strcpy(name, name0);
    }
};
Person _Cilk_shared someone;
char _Cilk_shared who[100];

int main(){
    strcpy(who, "Mike");
    _Cilk_offload someone.Set(who, 2);
}
MYO: Using Multiple Coprocessors

- To query the number of coprocessors:
  \[ \text{int nmics} = \_\_\_\text{Offload\_number\_of\_devices}() \]

- To specify which coprocessor \( n < \text{nmics} \) should do the computation:
  \[ \_\_\_\text{Cilk\_offload\_to}(n) \text{ } \text{func}(); \]

- Asynchronous offload:
  \[ \_\_\_\text{Cilk\_spawn\_Cilk\_offload\_to}(i) \text{ } \text{func}(); \]

Within \text{func()} also \text{OpenMP}, \text{pthreads} etc. can be used.
MYO: Multiple Offloads

```c
int _Cilk_shared *response;
void _Cilk_shared Respond(int _Cilk_shared & a) { a=1;}

response = (int _Cilk_shared *) _Offload_shared_malloc(n_d*sizeof(int));

int n_d = _Offload_number_of_devices();
response[0:n_d] = 0;

for (int i = 0; i < n_d; i++) {
    _Cilk_spawn _Cilk_offload_to(i) Respond(response[i]);
}
_Cilk_sync;
```

Needed only for asynchronous offload.
Offload: Using several MIC Coprocessors

- To query the number of coprocessors:
  
  ```c
  int nmics = __Offload_number_of_devices()
  ```

- To specify which coprocessor \( n < \) nmics should do the computation:
  
  ```c
  #pragma offload target(mic:n)
  ```

- If \( n > \) nmics then coprocessor \( (n \% nmics) \) is used

- **Important for:**
  - Asynchronous offloads
  - Coprocessor-Persistent data
## MYO Language Extensions

<table>
<thead>
<tr>
<th>Entity</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>int _Cilk_shared f(int x){...}</td>
<td>Executable code for both host and MIC; may be called from either side</td>
</tr>
<tr>
<td>Global variable</td>
<td>_Cilk_shared int x = 0</td>
<td>Visible on both sides</td>
</tr>
<tr>
<td>File/Function static</td>
<td>static _Cilk_shared int x</td>
<td>Visible on both sides, only to code within the file/function</td>
</tr>
<tr>
<td>Class</td>
<td>class _Cilk_shared x {...}</td>
<td>Class methods, members, and operators are available on both sides</td>
</tr>
<tr>
<td>Pointer to shared data</td>
<td>int _Cilk_shared *p</td>
<td>p is local (not shared), can point to shared data</td>
</tr>
<tr>
<td>A shared pointer</td>
<td>int * _Cilk_shared p</td>
<td>p is shared, should only point at shared data</td>
</tr>
<tr>
<td>Offloading a function call</td>
<td>x = _Cilk_offload func(y)</td>
<td>func executes on MIC if possible</td>
</tr>
<tr>
<td></td>
<td>x = _Cilk_offload_to(n) func</td>
<td>func must be executed on specified (n-th) MIC</td>
</tr>
<tr>
<td>Offloading asynchronously</td>
<td>_Cilk_spawn _Cilk_offload func(y)</td>
<td>Non-blocking offload</td>
</tr>
<tr>
<td>Offload a parallel for-loop</td>
<td>_Cilk_offload _Cilk_for(i=0; i&lt;N; i++) {...}</td>
<td>Loop executes in parallel on MIC</td>
</tr>
</tbody>
</table>
Intel Xeon Phi Programming Models: MPI
MPI on Hosts & MICs

<table>
<thead>
<tr>
<th>Pure MPI</th>
<th>Hybrid MPI/ OpenMP</th>
<th>Hybrid MPI/ OpenMP</th>
<th>MPI &amp; Offload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 multithreaded MPI-process per core</td>
<td>several multithreaded MPI-processes per core</td>
<td></td>
</tr>
</tbody>
</table>
• Default Module:
  – SuperMUC: mpi.ibm/1.3
  – SuperMIC: mpi.intel/5.0

• If you compile for MIC on SuperMUC login nodes use:
  – lu65fok@login07:~> module unload mpi.ibm
  – lu65fok@login07:~> module load mpi.intel
Important MPI environment variables

- Important Paths are already set by LRZ module, otherwise use:
  - . $ICC_BASE/bin/compilervars.sh intel64
  - . $MPI_BASE/bin64/mpivars.sh

- At LRZ the following MIC-specific environment variables are set per default on SuperMIC:
  - I_MPI_MIC=enable
  - I_MPI_HYDRA_BOOTSTRAP=ssh
  - I_MPI_FABRICS=shm:dapl
  - I_MPI_DAPL_PROVIDER_LIST=ofa-v2-mlx4_0-1,ofa-v2-scif0 (must be tuned)
## Invocation of the Intel MPI compiler

<table>
<thead>
<tr>
<th>Language</th>
<th>MPI Compiler</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>mpiicc</td>
<td>icc</td>
</tr>
<tr>
<td>C++</td>
<td>mpiicpc</td>
<td>icpc</td>
</tr>
<tr>
<td>Fortran</td>
<td>mpiifort</td>
<td>ifort</td>
</tr>
</tbody>
</table>
The following network fabrics are available for the Intel Xeon Phi coprocessor:

<table>
<thead>
<tr>
<th>Fabric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shm</td>
<td>Shared-memory</td>
</tr>
<tr>
<td>tcp</td>
<td>TCP/IP-capable network fabrics, such as Ethernet and InfiniBand (through IPoIB)</td>
</tr>
<tr>
<td>ofa</td>
<td>OFA-capable network fabric including InfiniBand (through OFED verbs)</td>
</tr>
<tr>
<td>dapl</td>
<td>DAPL–capable network fabrics, such as InfiniBand, iWarp, Dolphin, and XPMEM (through DAPL)</td>
</tr>
</tbody>
</table>
I_MPI_FABRICS

- The default can be changed by setting the I_MPI_FABRICS environment variable to I_MPI_FABRICS=<fabric> or I_MPI_FABRICS=<intra-node fabric>:<inter-nodes fabric>

- Intranode: Shared Memory, Internode: DAPL (Default on SuperMIC/MUC)
  - export I_MPI_FABRICS=shm:dapl

- Intranode: Shared Memory, Internode: TCP (Can be used in case of Infiniband problems)
  - export I_MPI_FABRICS=shm:tcp
Sample MPI Program

lu65fok@login12:~/tests> cat testmpi.c
#include <stdio.h>
#include <mpi.h>

int main (int argc, char* argv[]) {
    char hostname[100];
    int rank, size;

    MPI_Init (&argc, &argv);       /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);   /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size);   /* get number of processes */

    gethostname(hostname,100);
    printf( "Hello world from process %d of %d: host: %s\n", rank, size, hostname);
    MPI_Finalize();
    return 0;
}
MPI on hosts

- Compile for host using mpiicc / mpiifort:
  
  ```
  lu65fok@login12:~/tests> mpiicc testmpi.c -o testmpi-host
  ```

- Run 2 MPI tasks on host node i01r13a01

  ```
  lu65fok@login12:~/tests> mpiexec -n 2 -host i01r13a01 ./testmpi-host
  Hello world from process 0 of 2: host: i01r13a01
  Hello world from process 1 of 2: host: i01r13a01
  ```
MPI in native mode on 1 MIC

- Compile for MIC using mpiicc / mpiifort -mmic:
  lu65fok@login12:~/tests> mpiicc -mmic testmpi.c -o testmpi-mic

- Copy binary to MIC:
  lu65fok@login12:~/tests> scp testmpi-mic i01r13a01-mic0:

- Launch 2 MPI tasks from MIC node i01r13a01-mic0
  lu65fok@i01r13a04:~/tests> ssh i01r13a01-mic0
  [lu65fok@i01r13a01-mic0 ~]$ mpiexec -n 2 ./testmpi-mic
  Hello world from process 1 of 2: host: i01r13a01-mic0
  Hello world from process 0 of 2: host: i01r13a01-mic0
Do not mix up with mpicc and mpifort!!

lu65fok@login12:~/tests> mpicc -mmic testmpi.c -o testmpi-mic
/usr/lib64/gcc/x86_64-suse-linux/4.3/.../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpigf.so when searching for -lmpigf
/usr/lib64/gcc/x86_64-suse-linux/4.3/.../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpigf.a when searching for -lmpigf
/usr/lib64/gcc/x86_64-suse-linux/4.3/.../x86_64-suse-linux/bin/ld: cannot find -lmpigf
/usr/lib64/gcc/x86_64-suse-linux/4.3/.../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpi.so when searching for -lmpi
/usr/lib64/gcc/x86_64-suse-linux/4.3/.../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpi.a when searching for -lmpi
/usr/lib64/gcc/x86_64-suse-linux/4.3/.../x86_64-suse-linux/bin/ld: cannot find -lmpi
/usr/lib64/gcc/x86_64-suse-linux/4.3/.../x86_64-suse-linux/bin/ld: skipping incompatible
/lrz/sys/intel/mpi_41_3_048/mic/lib/libmpigi.a when searching for -lmpigi
/usr/lib64/gcc/x86_64-suse-linux/4.3/.../x86_64-suse-linux/bin/ld: cannot find -lmpigi
collect2: ld returned 1 exit status
MPI on 1 MIC

- Compile for MIC using mpiicc / mpiifort -mmic:
  
  ```bash
  lu65fok@login12:~/tests> mpiicc -mmic testmpi.c -o testmpi-mic
  ```

- Copy binary to MIC:
  
  ```bash
  lu65fok@login12:~/tests> scp testmpi-mic i01r13a01-mic0:
  ```

- Run 2 MPI tasks on MIC node i01r13a01-mic0
  
  ```bash
  lu65fok@i01r13a04:~/tests> mpiexec -n 2 -host i01r13a01-mic0
  ./home/lu65fok/testmpi-mic
  Hello world from process 1 of 2: host: i01r13a01-mic0
  Hello world from process 0 of 2: host: i01r13a01-mic0
  ```

**Full path needed!**
MPI on 2 MICs

- Compile for MIC using mpiicc / mpiifort -mmic:
  lu65fok@login12:~/tests> mpiicc -mmic testmpi.c -o testmpi-mic

- Copy binary to MICs:
  lu65fok@login12:~/tests> scp testmpi-mic i01r13a01-mic0:
  lu65fok@login12:~/tests> scp testmpi-mic i01r13a01-mic1:

- Run 2 MPI tasks on MIC node i01r13a01-mic0
  lu65fok@login12:~/tests> mpiexec -n 2 -perhost 1 -host
  i01r13a01-mic0,i01r13a01-mic1 .:/home/lu65fok/testmpi-mic
  Hello world from process 1 of 2: host: i01r13a01-mic1
  Hello world from process 0 of 2: host: i01r13a01-mic0
MPI on Host and 2 MICs attached to the host

```
lu65fok@login12:~/tests> mpirun -n 1 -host i01r13a01 ./testmpi-host : -n 1 -host i01r13a01-mic0 /home/lu65fok/testmpi-mic : -n 1 -host i01r13a01-mic1 /home/lu65fok/testmpi-mic

Hello world from process 0 of 3: host: i01r13a01
Hello world from process 2 of 3: host: i01r13a01-mic1
Hello world from process 1 of 3: host: i01r13a01-mic0
```
MPI on multiple Hosts & MICs

lu65fok@i01r13a01:~/tests> mpirun -n 1 -host i01r13a01 ./testmpi-host : -n 1 -host i01r13a01-mic0 /home/lu65fok/testmpi-mic : -n 1 -host i01r13a01-mic1 /home/lu65fok/testmpi-mic : -n 1 -host i01r13a02 ./testmpi-host : -n 1 -host i01r13a02-mic0 /home/lu65fok/testmpi-mic : -n 1 -host i01r13a02-mic1 /home/lu65fok/testmpi-mic

Hello world from process 3 of 6: host: i01r13a02
Hello world from process 0 of 6: host: i01r13a01
Hello world from process 2 of 6: host: i01r13a01-mic1
Hello world from process 5 of 6: host: i01r13a02-mic1
Hello world from process 1 of 6: host: i01r13a01-mic0
Hello world from process 4 of 6: host: i01r13a02-mic0
MPI Machine File

lu65fok@login12:~/tests> cat machinefile.txt
i01r13a01-mic0
i01r13a01-mic1
i01r13a02-mic0
i01r13a02-mic1

lu65fok@login12:~/tests> mpirun -n 4 -machinefile machinefile.txt /home/lu65fok/testmpi-mic
Hello world from process 3 of 4: host: i01r13a02-mic1
Hello world from process 2 of 4: host: i01r13a02-mic0
Hello world from process 1 of 4: host: i01r13a01-mic1
Hello world from process 0 of 4: host: i01r13a01-mic0
MPI Machine File

lu65fok@login12:~/tests> cat machinefile.txt
i01r13a01-mic0:2
i01r13a01-mic1
i01r13a02-mic0
i01r13a02-mic1

lu65fok@login12:~/tests> mpirun -n 4 -machinefile machinefile.txt /home/lu65fok/testmpi-mic
Hello world from process 3 of 4: host: i01r13a02-mic0
Hello world from process 0 of 4: host: i01r13a01-mic0
Hello world from process 2 of 4: host: i01r13a01-mic1
Hello world from process 1 of 4: host: i01r13a01-mic0

28/04/2015
#include <unistd.h>
#include <stdio.h>
#include <mpi.h>

int main (int argc, char* argv[]) {
    char hostname[100];
    int rank, size;
    MPI_Init (&argc, &argv);  /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);       /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size);        /* get number of processes */

    gethostname(hostname,100);

    #pragma offload target(mic)
    {
        char michostname[50];
        gethostname(michostname, 50);
        printf("MIC: I am %s and I have %ld logical cores. I was called by process %d of %d: host: %s \n", michostname,
                sysconf(_SC_NPROCESSORS_ONLN), rank, size, hostname);
    }

    printf("Hello world from process %d of %d: host: %s\n", rank, size, hostname);
    MPI_Finalize();
    return 0;
}
Offload from MPI Tasks using 1 host

lu65fok@login12:~/tests> mpiicc testmpioffload.c -o testmpioffload
lu65fok@login12:~/tests> mpirun -n 4 -host i01r13a01 ./testmpioffload
Hello world from process 3 of 4: host: i01r13a01
Hello world from process 1 of 4: host: i01r13a01
Hello world from process 0 of 4: host: i01r13a01
Hello world from process 2 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 3 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 0 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 1 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 2 of 4: host: i01r13a01
Offload from MPI Tasks using multiple hosts

lu65fok@login12:~/tests> mpirun -n 4 -perhost 2 -host i01r13a01,i01r13a02 ./testmpioffload
Hello world from process 2 of 4: host: i01r13a02
Hello world from process 0 of 4: host: i01r13a01
Hello world from process 3 of 4: host: i01r13a02
Hello world from process 1 of 4: host: i01r13a01
MIC: I am i01r13a02-mic0 and I have 240 logical cores. I was called by process 2 of 4: host: i01r13a02
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 1 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 0 of 4: host: i01r13a01
MIC: I am i01r13a02-mic0 and I have 240 logical cores. I was called by process 3 of 4: host: i01r13a02

28/04/2015 Intel Xeon Phi Programming
#pragma offload target(mic:rank%2)
{
    char michostname[50];
    gethostname(michostname, sizeof(michostname));
    printf("MIC: I am %s and I have %ld logical cores. I was called by
          process %d of %d: host: %s \n", michostname,
          sysconf(_SC_NPROCESSORS_ONLN), rank, size,
          hostname);
}

Offload from MPI Tasks: Using both MICs

lu65fok@login12:~/tests> mpirun -n 4 -perhost 2 -host i01r13a01,i01r13a02 ./testmpioffload
Hello world from process 0 of 4: host: i01r13a01
Hello world from process 2 of 4: host: i01r13a02
Hello world from process 3 of 4: host: i01r13a02
Hello world from process 1 of 4: host: i01r13a01
MIC: I am i01r13a02-mic1 and I have 240 logical cores. I was called by process 3 of 4: host: i01r13a02
MIC: I am i01r13a01-mic1 and I have 240 logical cores. I was called by process 1 of 4: host: i01r13a01
MIC: I am i01r13a01-mic0 and I have 240 logical cores. I was called by process 0 of 4: host: i01r13a01
MIC: I am i01r13a02-mic0 and I have 240 logical cores. I was called by process 2 of 4: host: i01r13a02
Lab: MPI
Password-less login to hosts:
cd ~/.ssh
cat id_rsa.pub >> authorized_keys
Intel Xeon Phi: Optimisation
The Intel MIC Architecture: VPU

Vector Processing Unit (VPU)

- The VPU includes the EMU (Extended Math Unit) and executes 16 single-precision floating point, 16 32-bit integer operations or 8 double-precision floating point operations per cycle. Each operation can be a FMA, giving 32 single-precision or 16 double-precision floating-point operations per cycle.

- Contains the vector register file: 32 512-bit wide registers per thread context, each register can hold 16 singles or 8 doubles.

- Most vector instructions have a 4-clock latency with a 1 clock throughput.
# MMX, SSE, AVX and IMCI Instruction Sets

<table>
<thead>
<tr>
<th>Instruction Set</th>
<th>Year &amp; Processor</th>
<th>SIMD Width</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMX</td>
<td>1997 Pentium</td>
<td>64-bit</td>
<td>8/16/32-bit Int.</td>
</tr>
<tr>
<td>SSE</td>
<td>1999 Pentium III</td>
<td>128-bit</td>
<td>32-bit SP FP</td>
</tr>
<tr>
<td>SSE2</td>
<td>2001 Pentium 4</td>
<td>128-bit</td>
<td>8-64-bit Int., SP&amp;DP FP</td>
</tr>
<tr>
<td>SSE3-SSE4.2</td>
<td>2004-2009</td>
<td>128-bit</td>
<td>Additional instructions</td>
</tr>
<tr>
<td>AVX</td>
<td>2011 Sandy-Bridge</td>
<td>256-bit</td>
<td>SP &amp; DP FP</td>
</tr>
<tr>
<td>AVX2</td>
<td>2013 Haswell</td>
<td>256-bit</td>
<td>int. &amp; additional instrcuct.</td>
</tr>
<tr>
<td>IMCI</td>
<td>2012 KNC</td>
<td>512-bit</td>
<td>32/64-bit Int., SP &amp; DP FP</td>
</tr>
</tbody>
</table>
SIMD Fused Multiply Add (FMA)

\[ \text{vfmad213ps source1, source2, source3} \]
IMCI Instruction Set

- IMCI: Initial Many-Core instruction set

IMCI is not SSE/AVX!

<table>
<thead>
<tr>
<th>IMCI Intrinsics</th>
<th>SSE2 Intrinsics</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (int i=0; i&lt;n; i+=4) { ___m128 Avec=_mm_load_ps(A+i); ___m128 Bvec=_mm_load_ps(B+i); Avec=_mm_add_ps(Avec, Bvec); _mm_store_ps(A+i, Avec); }</td>
<td>for (int i=0; i&lt;n; i+=16) { ___m512 Ave=}</td>
</tr>
</tbody>
</table>
Performance

- Sandy-Bridge-EP: 2 sockets × 8 cores @ 2.7 GHz.
- Xeon Phi: 60 cores @1.0 GHz.

- **# cycles/s:**
  - SandyBridge: 4.3E10 cycles/s.
  - Xeon Phi: 6.0E10 cycles /s.

- **DP FLOP/s:**
  - SandyBridge: 2 sockets × 8 cores × 2.7 GHz × 4 (SIMD) × 2 (ALUs) = 345.6 GFLOP/s
  - Xeon Phi: 60 cores × 1 GHz × 8 (SIMD) × 2 (FMA)= 960 GFLOP/s  Factor 2.7
Memory Bandwidth

- **Sandy-Bridge:**
  2 sockets × 4 memory channels × 6.4 GT/s × 2 bytes per channel = 102.4 GB/s

- **Xeon Phi:**
  8 memory controllers × 2 channels/controller × 6.0 GT/s × 4 bytes per channel = 384 GB/s.

- For complicated memory access patterns: memory latency / cache performance is important. Xeon Phi caches less powerful than Xeon caches (e.g. no L1 prefetcher etc.).

Factor 3.8
When is Xeon Phi expected to deliver better performance than the host:

1. **Bandwidth-bound code**: If memory access patterns are streamlined so that application is limited by memory bandwidth and not memory-latency bound.
2. **Compute-bound code**: high arithmetic intensity (# operations / memory transfer).
3. **Code should not be dominated by Host <-> MIC communication**: limited by slow PCIe v2 bandwidth of 6 GB/s.
Vectorization: Most important to get performance on Xeon Phi!!

- **#pragma ivdep**: Instructs the compiler to ignore assumed vector dependencies.
- **#pragma loop_count**: Specifies the iterations for the for loop.
- **#pragma novector**: Specifies that the loop should never be vectorized.
- **#pragma omp simd**: Transforms the loop into a loop that will be executed concurrently using Single Instruction Multiple Data (SIMD) instructions. (OpenMP 4.0)
#pragma vector

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>always</code></td>
<td>instructs the compiler to override any efficiency heuristic during the decision to vectorize or not, and vectorize non-unit strides or very unaligned memory accesses; controls the vectorization of the subsequent loop in the program; optionally takes the keyword assert</td>
</tr>
<tr>
<td><code>aligned</code></td>
<td>instructs the compiler to use aligned data movement instructions for all array references when vectorizing</td>
</tr>
<tr>
<td><code>unaligned</code></td>
<td>instructs the compiler to use unaligned data movement instructions for all array references when vectorizing</td>
</tr>
<tr>
<td><code>nontemporal</code></td>
<td>directs the compiler to use non-temporal (that is, streaming) stores on systems based on all supported architectures, unless otherwise specified; optionally takes a comma separated list of variables. On systems based on Intel® MIC Architecture, directs the compiler to generate clevict (cache-line-evict) instructions after the stores based on the non-temporal pragma when the compiler knows that the store addresses are aligned; optionally takes a comma separated list of variables.</td>
</tr>
<tr>
<td><code>temporal</code></td>
<td>directs the compiler to use temporal (that is, non-streaming) stores on systems based on all supported architectures, unless otherwise specified</td>
</tr>
<tr>
<td><code>vecremainder</code></td>
<td>instructs the compiler to vectorize the remainder loop when the original loop is vectorized</td>
</tr>
<tr>
<td><code>novecremainder</code></td>
<td>instructs the compiler not to vectorize the remainder loop when the original loop is vectorized</td>
</tr>
</tbody>
</table>
Example for vectorisation pragmas

```c
#pragma offload target(mic) in(a,b:length(n*n)) inout(c:length(n*n)) {
#pragma omp parallel for
for( i = 0; i < n; i++ ) {
    for( k = 0; k < n; k++ ) {
        #pragma vector aligned
        #pragma ivdep
        for( j = 0; j < n; j++ ) {
            //c[i][j] = c[i][j] + a[i][k]*b[k][j];
            c[i*n+j] = c[i*n+j] + a[i*n+k]*b[k*n+j];
        }
    }
}
```

28/04/2015 Intel Xeon Phi Programming
#pragma simd

- The simd pragma is used to guide the compiler to vectorize more loops. Vectorization using the simd pragma complements (but does not replace) the fully automatic approach.

- Without explicit `vectorlength()` and `vectorlengthfor()` clauses, compiler will choose a vectorlength using its own cost model. Misclassification of variables into `private`, `firstprivate`, `lastprivate`, `linear`, and `reduction`, or lack of appropriate classification of variables may lead to unintended consequences such as runtime failures and/or incorrect result.
#pragma simd

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

Function uses too many unknown pointers for the compiler's automatic runtime independence check optimization to kick-in
Thread Affinity

- Pinning Threads is important!
- `export KMP_AFFINITY="granularity=thread,x"`  
  `x=compact, scatter, balanced`
- See Intel compiler Documentation.
Data alignment

- Prerequisite for successful use of the SIMD units.

- A pointer p is said to address a memory location aligned on an n-byte boundary if ((size_t)p%n==0).

- Memory address should a multiple of the vector register width in bytes, i.e.
  - SSE2: 16-Byte alignment
  - AVX: 32-Byte alignment
  - MIC: 64-Byte alignment
Data alignment

- **Data alignment on the stack:**
  - `__declspec(align(64)) double data[N];` (ICC)
  - `double data[N] __attribute__((aligned(64)));` (ICC, GCC)

- **Data alignment on the heap (ICC)**
  - `_mm_malloc/free` functions (ICC)
    ```
    #include <malloc.h>
    double*A = (double*)_mm_malloc(N*sizeof(double), 64);
    _mm_free(A);
    ```
  - `posix_memalign` (ICC, GCC)
    ```
    ok=posix_memalign((void**)&a, 64, n*n*sizeof(double));
    ```
Prefetching

- **Hardware Prefetching:**
  - **Intel Xeon processors:** L1 and L2 hardware prefetchers
  - **Intel Xeon Phi:** only L2 hardware prefetcher

- **Software Prefetching:**
  - Instructions that request that a cache line is fetched from memory in cache.
  - Does not stall execution.
  - **Prefetch distance** = time between the prefetch instruction and the instruction using the data
Prefetching

- If software prefetches are doing a good job, then hardware prefetching does not kick in.
- In several workloads (such as stream), maximal software prefetching gives the best performance.
- Any references not prefetched by compiler may get prefetched by hardware.
- Details: Rakesh Krishnayer
Summary

- Concerning the ease of use and the programmability, Intel Xeon Phi is a promising hardware architecture compared to other accelerators like GPGPUs, FPGAs or former CELL processors or ClearSpeed cards.
- Codes using MPI, OpenMP or MKL etc. can be quickly ported. Some MKL routines have been highly optimised for the MIC.
- Due to the large SIMD width of 64 Bytes, vectorisation is even more important for the MIC architecture than for Intel Xeon based systems.
- It is extremely simple to get a code running on Intel Xeon Phi, but getting performance out of the chip in most cases needs manual tuning of the code due to failing auto-vectorisation.
- MIC programming enforces programmer to think about SIMD vectorisation
  → Performance on current /future Xeon based systems also much better with MIC-optimised code.
References & Credits

- Books:
    http://lotsofcores.com
  - Parallel Programming and Optimization with Intel Xeon Phi Coprocessors, Colfax 2013. 
- Intel Xeon Phi Programming, Training material, CAPS
- Intel Training Material and Webinars, credits to M. Klemm
- V. Weinberg (Editor) et al., Best Practice Guide - Intel Xeon Phi, http://www.prace-project.eu/Best-Practice-Guide-Intel-Xeon-Phi-HTML and references therein