OpenMP 4.0 Acceleration

TACC IXPUG14

Austin, TX

Eric Stotzer, Ph.D.

Abstract

Hardware and software advances in DSP, GPU, MIC, ARM and FPGA technologies have accelerated the need for a common many-threaded model for these accelerators. The OpenMP Language Committee has also accelerated its pace and is finalizing features for the 4.1 release that will provide a common threading model for many-core technologies.

Insights into some of the design decisions that went into the OpenMP accelerator model will be presented. Also, a preview of the OpenMP accelerator sub-committee's future releases for the OpenMP specificatio will be outlined and discussed.

Why Texas Instruments?

WHY ME?

High Performance Embedded Computing

DVR / NVR & smart camera

Networking

Mission critical systems

Medical imaging

Video and audio infrastructure

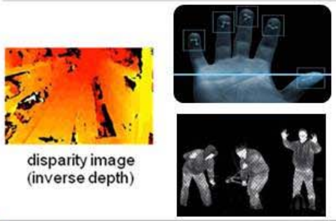
High-performance and cloud computing

Portable mobile radio

Industrial imaging

Home AVR and automotive audio

Analytics



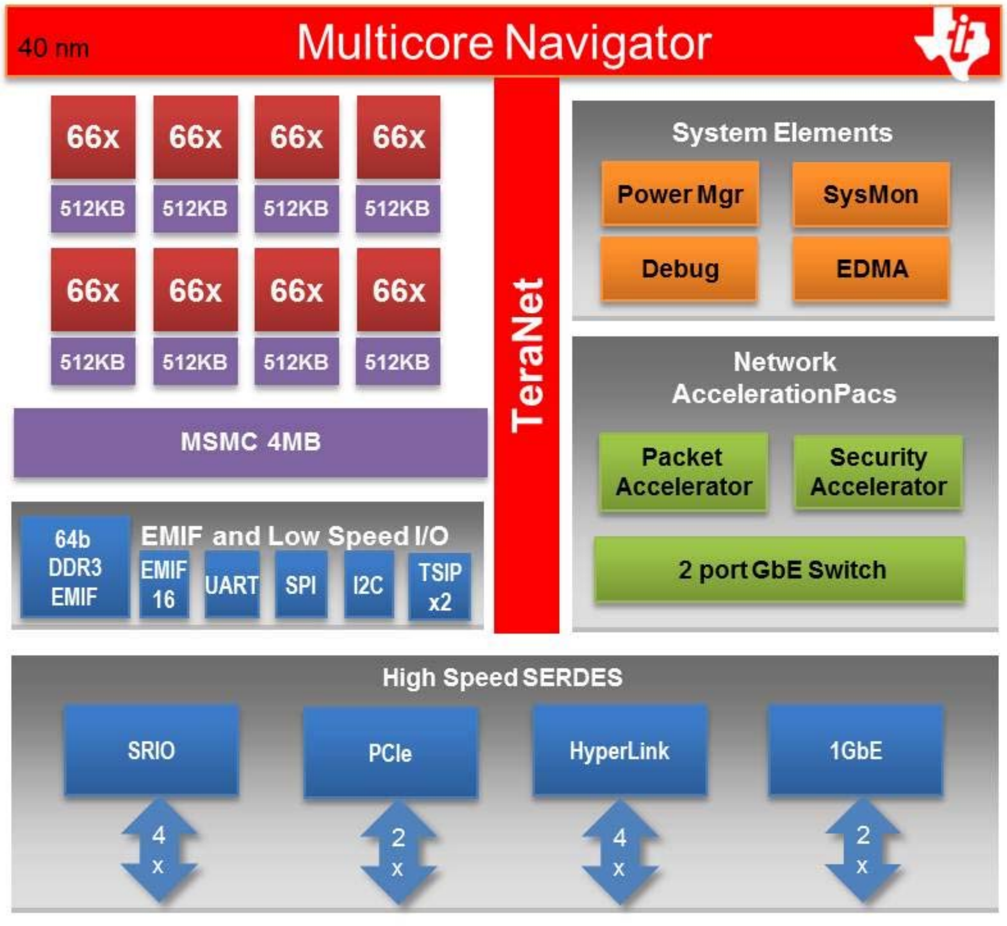
Wireless testers

Industrial control

media processing | *computing* | *radar & communications* | *industrial electronics*

Keystone I: C6678 SoC

- Eight 8 C66x cores
- Each with 32k L1P, 32k L1D, 512k L2
- 1 to 1.25 GHz
- 320 GMACS
- 160 SP GFLOPS
- 512 KB/Core of local L2
- 4MB Multicore Shared Memory (MSMC)
- Multicore Navigator (8k HW queues) and TeraNet
- Serial-RapidIO, PCIe-II, Ethernet, 1xHyperlink



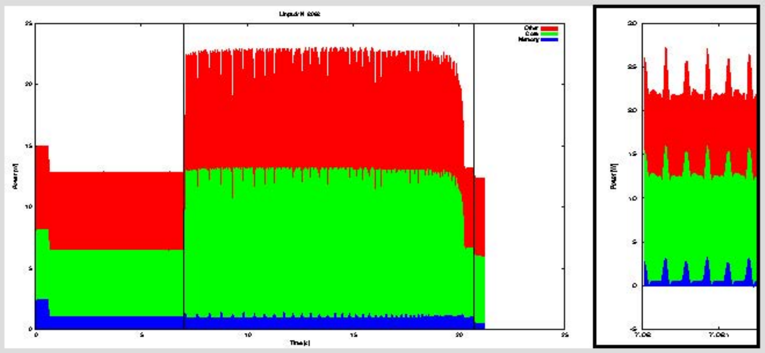
24mm x 24mm package

Energy Efficiency

LINPACK running on C6678 achieves 25.6 Gflops, ~2.1 Gflops/W

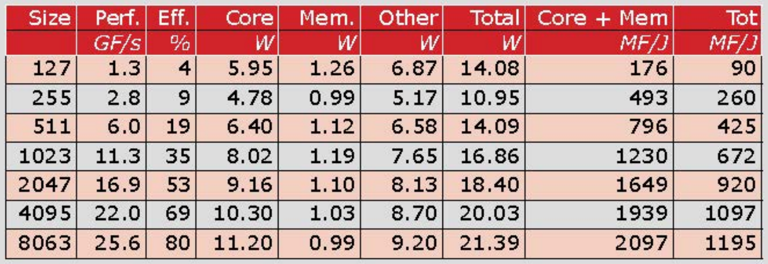
PRACE First Implementation Project, Grant RI-261557, Final Report on Prototypes Evaluation. Lennart Johnsson, Gilbert Netzer, SNIC/KTH, 3/29/2013.

Linpack Power Profile



The plot shows the power consumption over time during a single execution of the Linpack benchmark code. Blue shows memory power, green is added power fed to the DSP and red other module consumers stacked atop. The vertical lines denote the timed section of the code. Distinct phases of execution can be seen, for instance the serial back-substitution at the end of the run. A zoom in also reveals the power peaks caused by DMA block copies to and from the main memory.

DSP Linpack Energy Efficiency

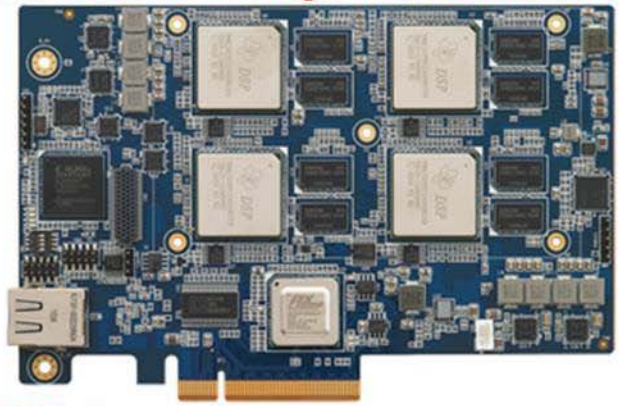


Size	Perf.	Eff.	Core	Mem.	Other	Total	Core + Mem	Tot
	GF/s	%	W	W	W	W	MF/J	MF/J
127	1.3	4	5.95	1.26	6.87	14.08	176	90
255	2.8	9	4.78	0.99	5.17	10.95	493	260
511	6.0	19	6.40	1.12	6.58	14.09	796	425
1023	11.3	35	8.02	1.19	7.65	16.86	1230	672
2047	16.9	53	9.16	1.10	8.13	18.40	1649	920
4095	22.0	69	10.30	1.03	8.70	20.03	1939	1097
8063	25.6	80	11.20	0.99	9.20	21.39	2097	1195

The table shows the power and energy consumption of the major components of the C6678 DSP EVM. Core refers to the C6678 DSP SoC excluding I/O power. Mem is the DDR3 memory subsytem. The 5-9 watts of "Other" power is to a large part, except for about 1.5 W DC converter losses, consumed by debugging and unused hardware features that would not be present in an HPC server node. Therefore the "Core + Mem" power is a good estimate for the energy efficiency of an HPC server node. The values for small problem sizes show deviations due to various parts of the benchmark not being executed. The outmost loop step size is 128 columns, another breakpoint occurs at 1024.

High Density COTS boards

DSPC-8681 ½ length PCIe card - 54Watts



DSPC-8682 PCIe Full-Length Card - 110Watts

DSPC-8682 ATCA blade 350Watts

Keystone II: 66AK2H12/06 SoC

C66x Fixed or Floating Point DSP

- 4x/8x 66x DSP cores up to 1.4GHz
- 2x/4x Cotex ARM A15
- 1MB of local L2 cache RAM per C66 DSP core
- 4MB shared across all ARM

Large on chip and off chip memory

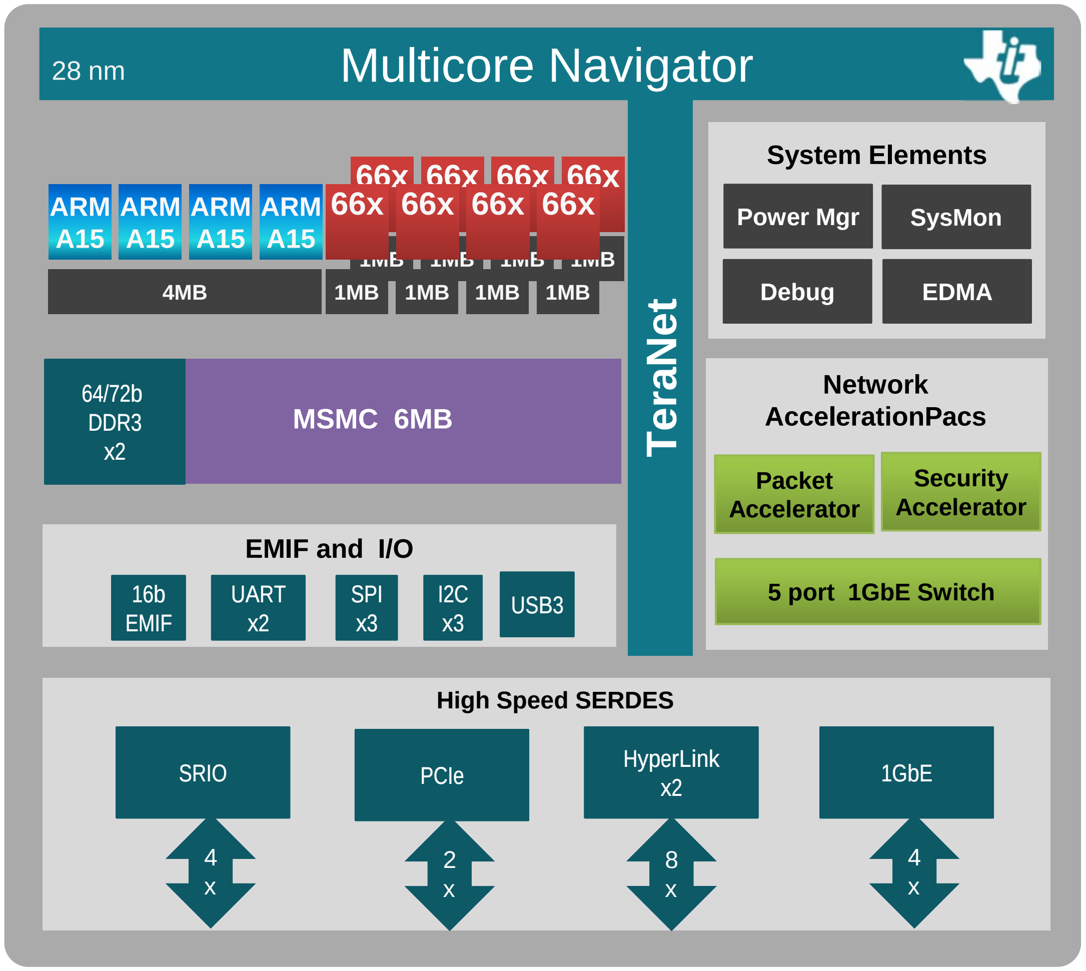
- Multicore Shared Memory Controller provides low latency & high bandwidth memory access
- 6MB Shared L2 on-chip
- 2 x 72 bit DDR3, 72-bit (with ECC), 10GB total addressable, DIMM support (4 ranks total)

KeyStone multicore architecture and acceleration

- Multicore Navigator, TeraNet, HyperLink
- 1GbE Network coprocessor (IPv4/IPv6)
- Crypto Engine (IPSec, SRTP)

Peripherals

- 4 Port 1G Layer 2 Ethernet Switch
- 2x PCIe, 1x4 SRIO 2.1, EMIF16, USB 3.0 UARTx2, SPI, I^{2}C
- 15-25W depending upon DSP cores, speed, temp & other factors



40mm x 40mm package

Available HPC Platforms

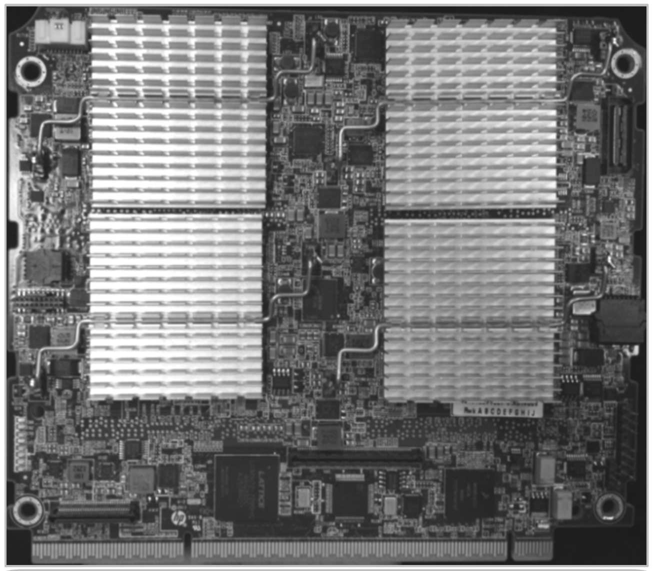
nCore BrownDwarf

nCore
BROWNDWARF
BrownDwarf Y-Class System
ARM+DSP Supercomputer
The nCore Y-Class supercomputer combines unprecedented low power computational performance with a

"The BrownDwarf Y-Class system is an incredibly important milestone in HPC system development. -Working in close collaboration with TI, IDT and our hardware partner Prodrive, we have successfully established a new class of energy efficient supercomputers designed to fulfill the demands of a wide range of scientific, technical and commercial applications. We are very excited to be launching the most capable energy efficient supercomputer available. The innovative design of the BrownDwarf Y-Class system has resulted in a network fabric that far exceeds the latency and power efficiencies of traditional supercomputing systems based on x86 and Infiniband or Ethernet systems.- By utilizing existing programming models and toolsets, the BrownDwarf Y-Class supercomputer is a disruptive force in HPC as it leapfrogs a number of the supercomputing incumbents."

-- Ian Lintault, Managing Director, nCore HPC

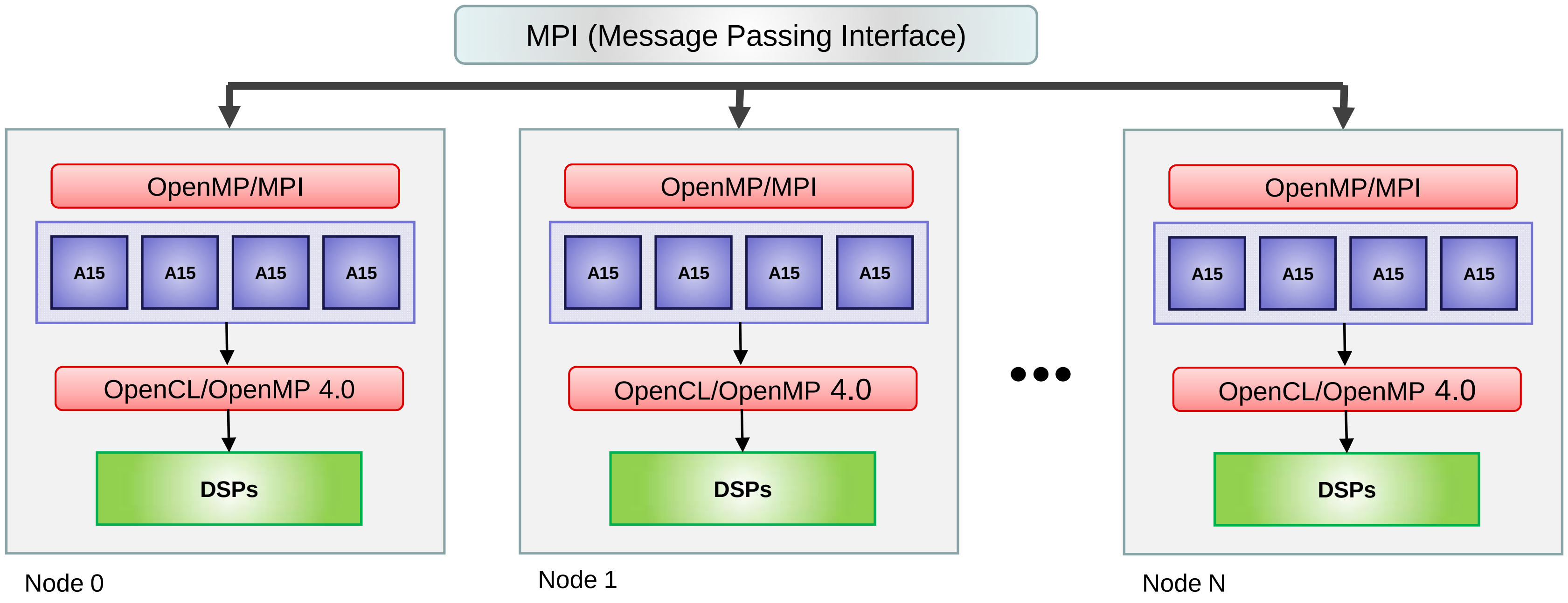
HP Moonshot



"As a partner in HP's Moonshot ecosystem dedicated to the rapid development of new Moonshot servers, we believe TI's KeyStone design will provide new capabilities across multiple disciplines to accelerate the pace of telecommunication innovations and geological exploration.**"**

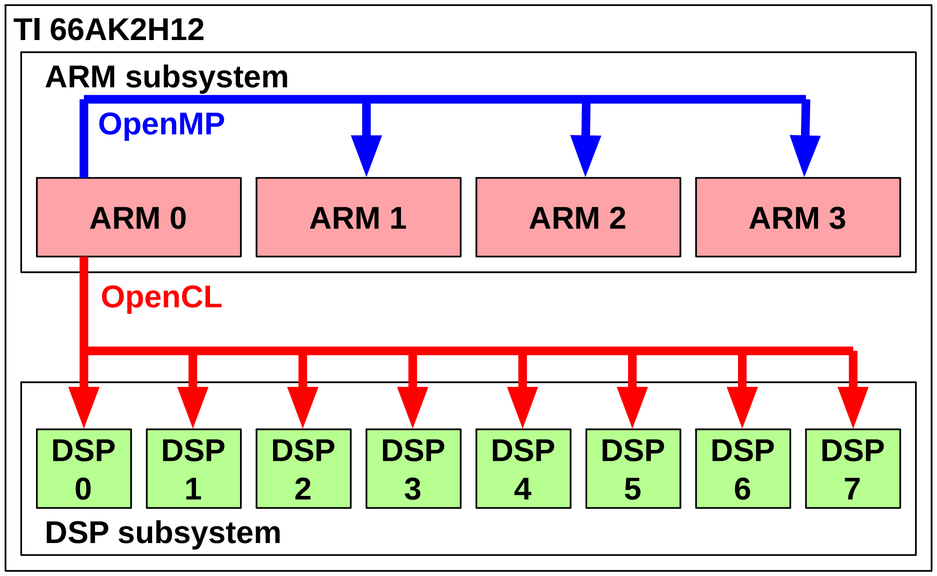
--- Paul Santeler, vice president and general manager, Hyperscale Business, HP

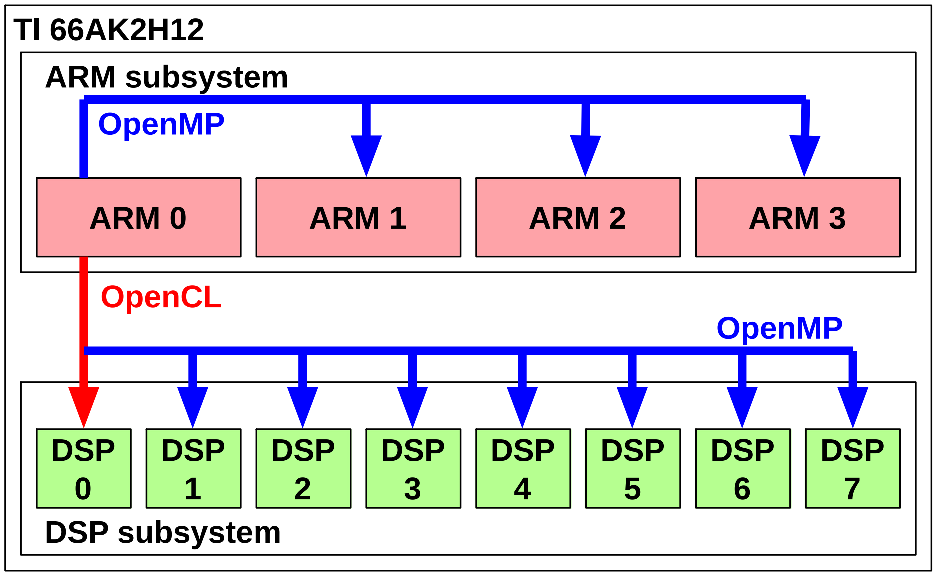
Heterogeneous Multicore Programming



- Within a node, OpenCL™ or OpenMP® 4.0 can be used to program heterogeneous compute cores
- Across nodes, MPI is used to partition the application and manage program execution, data transfer and synchronization

ARM + OpenCL DSP Acceleration





Data parallel
- A kernel is enqueued
- OpenCL divides into N workgroups
- Each workgroup is assigned a core
- After all workgroups finish a new kernel can be dispatched

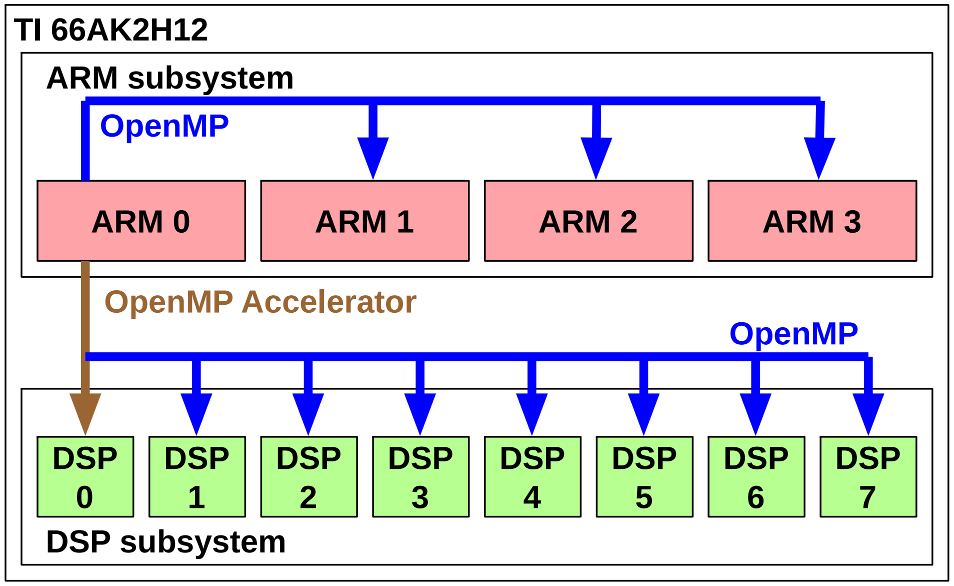
Task parallel
- A task is enqueued
- OpenCL dispatches tasks to cores
- OpenCL can accept and dispatch more tasks asynchronously

OpenCL + OpenMP regions
- A task is enqueued
- OpenCL dispatches the task to DSP 0
- Tasks can use additional DSP cores by entering OpenMP regions
- A task completes before another task is dispatched
- Note: This is a TI extension

Example use
- Want to call existing OpenMP based DSP code from the ARM

ARM + OpenMP 4.0

```
// OpenMP Accelerator vector add
// OpenMP for loop parallelization
void ompVectorAdd(int   N,
                  float *a,
                  float *b,
                  float *c)
{
  #pragma omp target           \
  map(to:   N, a[0:N], b[0:N]) \
  map(from: c[0:N])
  {
    int i;
    #pragma omp parallel for
    for (i = 0; i < N; i++)
      c[i] = a[i] + b[i];
  }
}
```



Data movement
- **`to`** copies variables from the ARM memory to the DSP memory
- **`from`** copies variables from the DSP memory to the ARM memory
- TI provides special **`alloc`** and **`free`** functions to allocate DSP memory such that copies are not needed

Calling existing DSP code from the ARM
- Wrapping existing DSP functions with OpenMP Accelerator code is straightforward

Support for Heterogeneous Compute Nodes

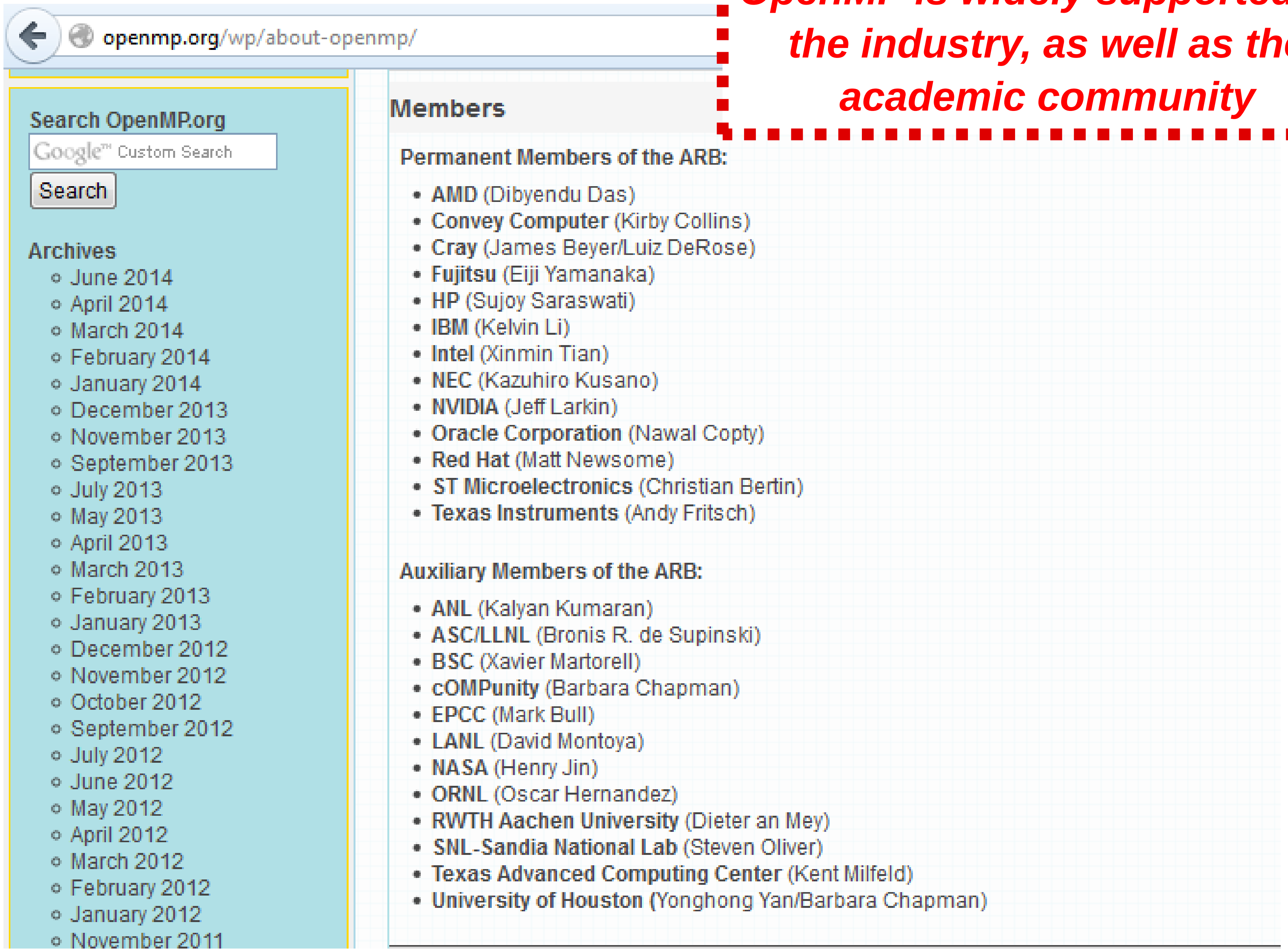
OPENMP 4.0

What is OpenMP?

- De-facto standard Application Programming Interface (API) to write shared memory parallel applications in C, C++, and Fortran
- Consists of Compiler Directives, Runtime routines and Environment variables
- Specification maintained by the OpenMP Architecture Review Board (http://www.openmp.org)
- New ARB mission statement:

  "The OpenMP ARB mission is to standardize directive-based multi-language high-level parallelism that is performant, productive and portable."
- **Version 4.0 has been released July 2013**

OpenMP is widely supported by the industry, as well as the academic community



openmp.org/wp/about-openmp/

Search OpenMP.org

Google™ Custom Search

Search

Archives

- June 2014
- April 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- September 2013
- July 2013
- May 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- October 2012
- September 2012
- July 2012
- June 2012
- May 2012
- April 2012
- March 2012
- February 2012
- January 2012
- November 2011

Members

Permanent Members of the ARB:

- **AMD** (Dibyendu Das)
- **Convey Computer** (Kirby Collins)
- **Cray** (James Beyer/Luiz DeRose)
- **Fujitsu** (Eiji Yamanaka)
- **HP** (Sujoy Saraswati)
- **IBM** (Kelvin Li)
- **Intel** (Xinmin Tian)
- **NEC** (Kazuhiro Kusano)
- **NVIDIA** (Jeff Larkin)
- **Oracle Corporation** (Nawal Copty)
- **Red Hat** (Matt Newsome)
- **ST Microelectronics** (Christian Bertin)
- **Texas Instruments** (Andy Fritsch)

Auxiliary Members of the ARB:

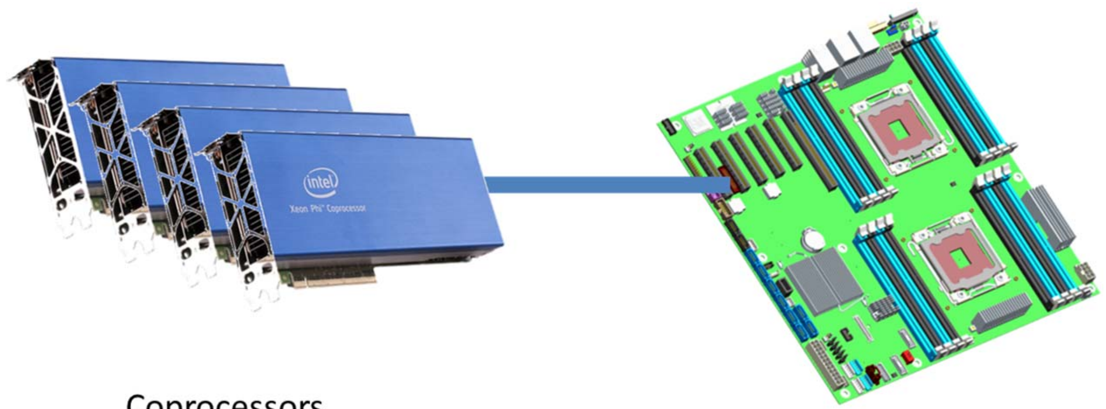
- **ANL** (Kalyan Kumaran)
- **ASC/LLNL** (Bronis R. de Supinski)
- **BSC** (Xavier Martorell)
- **cOMPunity** (Barbara Chapman)
- **EPCC** (Mark Bull)
- **LANL** (David Montoya)
- **NASA** (Henry Jin)
- **ORNL** (Oscar Hernandez)
- **RWTH Aachen University** (Dieter an Mey)
- **SNL-Sandia National Lab** (Steven Oliver)
- **Texas Advanced Computing Center** (Kent Milfeld)
- **University of Houston** (Yonghong Yan/Barbara Chapman)

New in OpenMP 4.0

- Support for accelerators (or heterogeneous devices)
- Thread affinity support
- SIMD support for vectorization
- Thread cancellation
- Fortran 2003 support
- Extended support for
  - Tasking (groups, dependencies, abort)
  - Reductions (i.e. User Defined Reductions)
  - Atomics (sequential consistency)

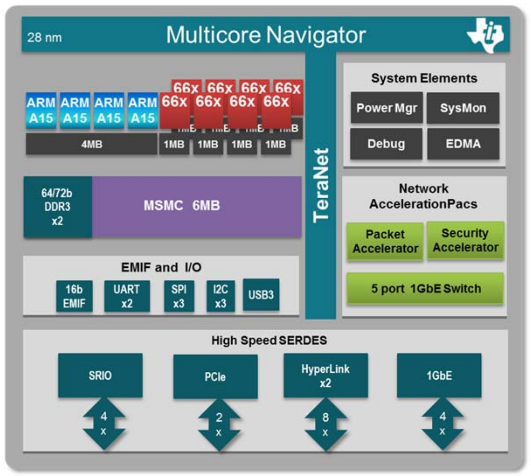
Heterogeneous Device model

- OpenMP 4.0 supports accelerators/coprocessors
- Device model:
  - One host
  - Multiple accelerators/coprocessors of the same kind



Coprocessors

Host



Heterogeneous SoC

Terminology

- Device:
  an implementation-defined (logical) execution unit
- Mapped variable:
  An original variable in a (host) data environment with a *corresponding variable* in a device data environment
- Mappable type:
  A type that is amenable for mapped variables.
  (Bitwise copyable plus additional restrictions.)
- Device data environment:
  Data environment as defined by `target data` or `target` constructs

The execution model is host-centric such that the host device offloads **target** regions to target devices.

OpenMP 4.0 Device Constructs

- Execute code on a target device
  - **omp target** *[clause[[,] clause],…]*
    structured-block
  - **omp declare target**
    [function-definitions-or-declarations]
- Map variables to a target device
  - **map** *([map-type:] list) // map clause*
    map-type := ***alloc*** | ***tofrom*** | ***to*** | ***from***
  - **omp target data** *[clause[[,] clause],…]*
    structured-block
  - **omp target update** *[clause[[,] clause],…]*
  - **omp declare target**
    [variable-definitions-or-declarations]
- Workshare for acceleration
  - **omp teams** *[clause[[,] clause],…]*
    structured-block
  - **omp distribute** *[clause[[,] clause],…]*
    for-loops

Device Runtime Support

- Runtime support routines
  - **`void omp_set_default_device(int`** ***`dev_num`*** **`)`**
  - **`int omp_get_default_device(void)`**
  - **`int omp_get_num_devices(void);`**
  - **`int omp_get_num_teams(void)`**
  - **`int omp_get_team_num(void);`**
  - **`Int omp_is_initial_device(void);`**
- Environment variable
  - Control default device through **`OMP_DEFAULT_DEVICE`**
  - Accepts a non-negative integer value

`target` Construct Example

- Use target construct to
  - Transfer control from the host to the device
  - Establish a device data environment (if not yet done)
- Host thread waits until offloaded region completed
  - Use other OpenMP constructs for asynchronicity

```
#pragma omp target map(to:b[0:count]) map(to:c,d) map(from:a[0:count])
  {
#pragma omp parallel for
    for (i=0; i<count; i++) {
      a[i] = b[i] * c + d;
    }
  }
```

host

target

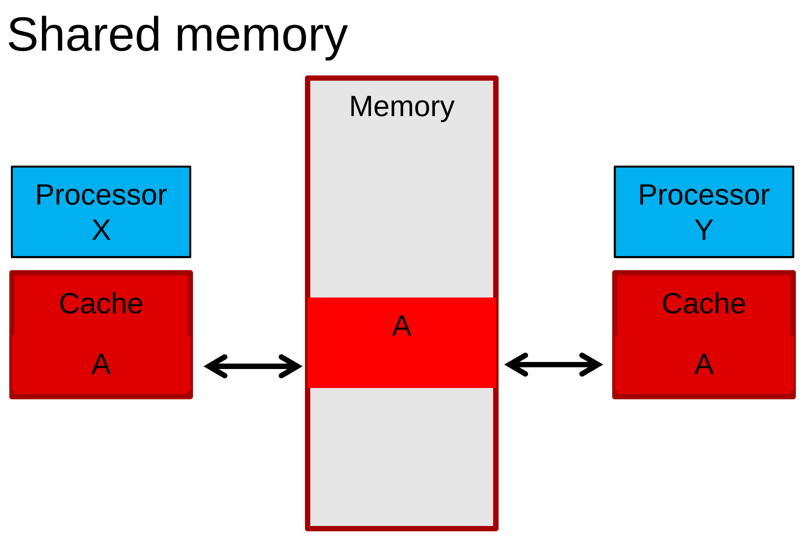
host

target data Construct Example

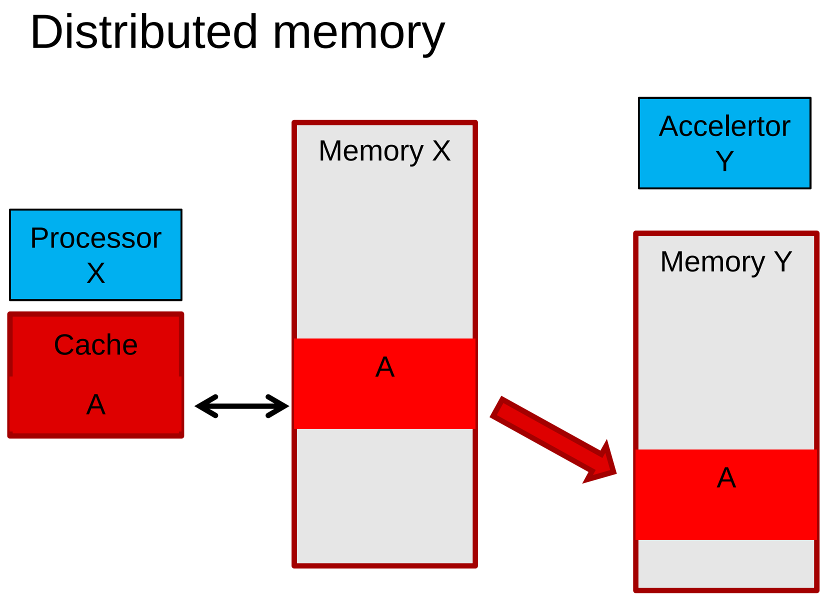
```
extern void init(float*, float*, int);
extern void init_again(float*, float*, int);
extern void output(float*, int);

void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;

   init(v1, v2, N);

   #pragma omp target data map(from: p[0:N])
   {
      #pragma omp target map(to: v1[:N], v2[:N])
      #pragma omp parallel for
      for (i=0; i<N; i++)
          p[i] = v1[i] * v2[i];

      init_again(v1, v2, N);

      #pragma omp target map(to: v1[:N], v2[:N])
      #pragma omp parallel for
      for (i=0; i<N; i++)
          p[i] = p[i] + (v1[i] * v2[i]);
   }

   output(p, N);
}
```

- The `target data` construct creates a *device data environment* and encloses `target` regions, which have their own device data environments.
- The device data environment of the `target data` region is inherited by the device data environment of an enclosed `target` region.
- The `target data` construct is used to create variables that will persist throughout the `target data` region.
- v1 and v2 are mapped at each `target` construct.
- Instead of mapping the variable p twice, once at each `target` construct, p is mapped once by the `target data` construct.

Data mapping: shared or distributed memory



- The corresponding variable in the device data environment *may* share storage with the original variable.
- Writes to the corresponding variable may alter the value of the original variable.



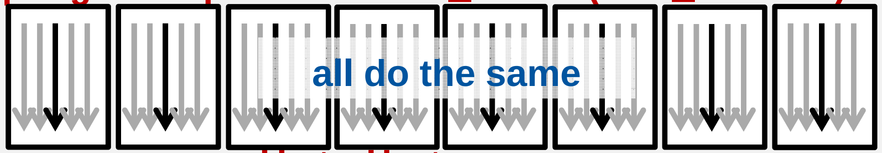
Terminology

- League:
  the set of threads teams created by a `teams` construct
- Contention group:
  threads of a team in a league and their descendant threads

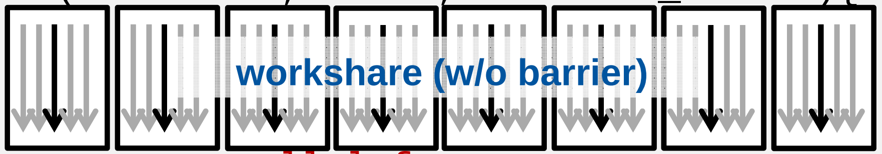
The **teams** construct creates a *league of thread teams* where the master thread of each team executes the region.

teams and distribute Constructs Example

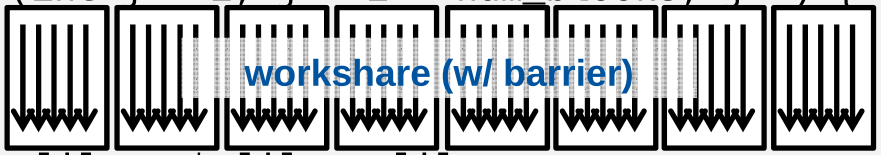
```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
{
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) thread_limit(bsize)
```



```
#pragma omp distribute
  for (int i = 0; i < n; i += num_blocks){
```



```
#pragma omp parallel for
    for (int j = i; j < i + num_blocks; j++) {
```



```
        y[j] = a*x[j] + y[j];
  } }
} free(x); free(y); return 0; }
```

distribute parallel for Construct Example

- SAXPY: Combined Constructs

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target map(to:x[0:n]) map(tofrom:y)
  {
#pragma omp teams num_teams(num_blocks) thread_limit(bsize)
#pragma omp distribute parallel for
    for (int i = 0; i < n; ++i){
      y[i] = a*x[i] + y[i];
    }
  }

  free(x); free(y); return 0;
}
```

OpenMP 4.0 Capabilities

Feature	**OpenACC**	**OpenMP 4.0**
Support for C and C++, Fortran	✔	✔
Support single code base of hetero-machine	✔	✔
Overlap communication and computation	✔	✔
Interoperate with MPI	✔	✔
Interoperate with OpenMP		✔
Offload to GPU	✔	✔
Offload to Intel Xeon Phi Coprocessor		✔
Ability to support all accelerators		✔
Ability to support all GPUs		✔
Ability to support all co-processors		✔
Support for nested parallelism		✔
User-managed memory consistency	✔	✔
Multiple vendor support	✔	✔
Support for dynamic dispatch		✔
Parallel on/off separate from offload		✔
Intel compiler support		2013
Broad standards body approval		✔

What are we working on in the OpenMP accelerator sub-committee?

OPENMP 4.1 AND BEYOND

Plan for OpenMP specifications

- OpenMP Tools Interface Technical Report
  - Released in March 2014
  - Working towards adoption in 5.0 (maybe 4.1)
- OpenMP 4.1 Technical Report
  - Changes adopted in time frame of SC14
  - Major focus will be device construct refinements
  - Provide clear guidance to begin 4.1 implementations
- OpenMP 4.1
  - Clarifications, refinements and minor extensions to existing specification
  - Do not break existing code
  - Minimal implementation burden beyond 4.0
  - Targeting release for SC15
- OpenMP 5.0
  - Address several major open issues for OpenMP
  - Expect less significant advance than 4.0 from 3.1/3.0
  - Do not break existing code unnecessarily
  - Targeting release for SC17 (somewhat ambitious)

Refinements to device constructs are the most significant 4.1 plans

- Refinements of combined clauses
  - Addition of even more combined constructs
  - Specifying overlapping clauses on combined constructs
- Asynchronous execution of target regions
- Unstructured data mapping
- Link clause/linkable support
- Multiple device types
- Deep copy/map/serialization for map
- Update for map even if present
- Providing device-specific environment variables

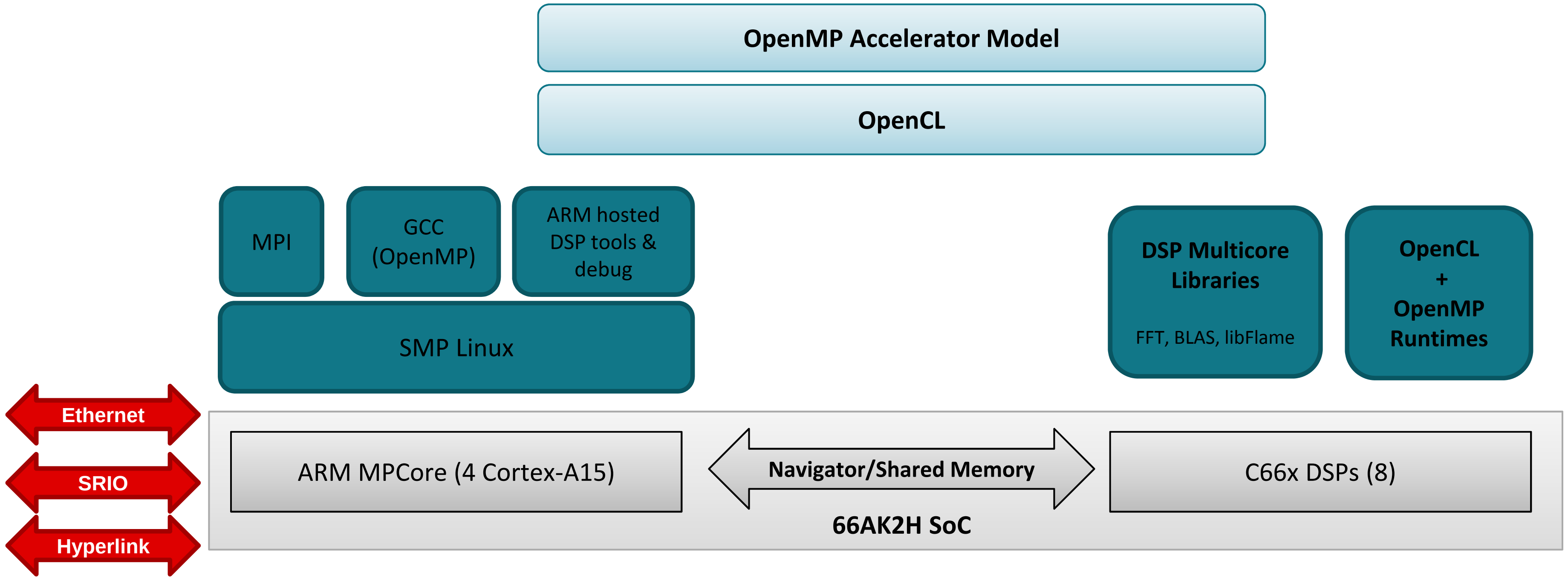
Want to learn more?

- Attend our advanced OpenMP programming tutorial at SC’14
- Attend IWOMP’14 in Salvador, Brazil, Sept. 2014.
- Join the ARB!

Never enough time…

BACKUP

Tooling for 66AK2H



- A node is a 66AK2H SoC
- OpenMP Accelerator model or OpenCL for offloading computation from ARMs to DSPs on a single node
- MPI on ARM to communicate across nodes (multiple transports supported)

`target` Construct

- Transfer control from the host to the device
- Syntax (C/C++)

```
#pragma omp target [clause[[,] clause],…]
structured-block
```

- Syntax (Fortran)

```
!$omp target [clause[[,] clause],…]
structured-block
!$omp end target
```

- Clauses

```
device(scalar-integer-expression)
map(alloc | to | from | tofrom: list)
if(scalar-expr)
```

`target declare` Construct

- Declare one or more functions to also be compiled for the target device

- Syntax (C/C++):

```
#pragma omp declare target
    [function-definitions-or-declarations]
#pragma omp end declare target
```

- Syntax (Fortran):

```
 !$omp declare target [(proc-name-list | list)]
```

target data Construct

- Create a device data environment
- Syntax (C/C++)
  #pragma omp target data *[clause[[,] clause],…]*
  structured-block
- Syntax (Fortran)
  !$omp target data *[clause[[,] clause],…]*
  structured-block
  !$omp end target data
- Clauses
  device(*scalar-integer-expression*)
  map(alloc | to | from | tofrom: *list*)
  if(*scalar-expr*)

`target update` Construct

- Issue data transfers between host and devices
- Syntax (C/C++)
  `#pragma omp target update` *`[clause[[,] clause],…]`*
- Syntax (Fortran)
  `!$omp target update` *`[clause[[,] clause],…]`*
- Clauses
  `device``(`*`scalar-integer-expression`*`)`
  `to``(`*`list`*`)`
  `from``(`*`list`*`)`
  `if``(`*`scalar-expr`*`)`

teams Construct

- Syntax (C/C++):
  #pragma omp teams *[clause[[,] clause],…]*
  structured-block
- Syntax (Fortran):
  !$omp teams *[clause[[,] clause],…]*
  structured-block
- Clauses
  num_teams(*integer-expression*)
  thread_limit(*integer-expression*)
  default(shared | none)
  private(*list*), firstprivate(*list*)
  shared(*list*), reduction(*operator* : *list*)

`distribute` Construct

- Syntax (C/C++):
  `#pragma omp distribute` *`[clause[[,] clause],…]`*
  `for-loops`
- Syntax (Fortran):
  `!$omp teams` *`[clause[[,] clause],…]`*
  `do-loops`
- Clauses
  `private``(`*`list`*`)`
  `firstprivate``(`*`list`*`)`
  `collapse``(`*`n`*`)`
  `dist_schedule``(`*`kind[, chunk_size]`*`)`

`if` Clause Example

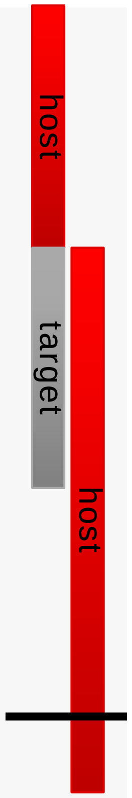
```
#define THRESHOLD1 1000000
#define THRESHOLD2 1000

extern void init(float*, float*, int);
extern void output(float*, int);

void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;
   init(v1, v2, N);

   #pragma omp target if(N>THRESHOLD1) \\
          map(to: v1[0:N], v2[:N]) map(from: p[0:N])
   #pragma omp parallel for if(N>THRESHOLD2)
   for (i=0; i<N; i++)
     p[i] = v1[i] * v2[i];
   output(p, N);
}
```

- The `if` clause on the `target` construct indicates that if the variable N is smaller than a given threshold, then the `target` region will be executed by the host device.
- The `if` clause on the `parallel` construct indicates that if the variable N is smaller than a second threshold then the parallel region is inactive.

Asynchronous Offloading

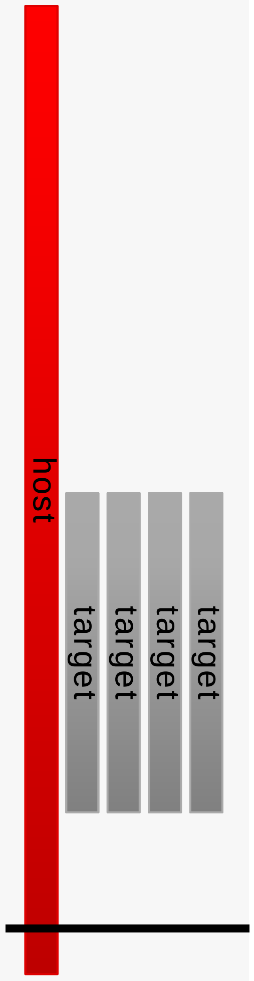
- Use existing OpenMP features to implement asynchronous offloads.

```
#pragma omp parallel sections
{
#pragma omp task
  {
#pragma omp target map(to:input[:N]) map(from:result[:N])
#pragma omp parallel for
    for (i=0; i<N; i++) {
      result[i] = some_computation(input[i], i);
    }
  }
#pragma omp task
  {
    do_something_important_on_host();
  }
#pragma omp taskwait
}
```



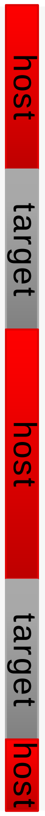
device Clause Example

```
int num_dev = omp_get_num_devices();
int chunksz = length / num_dev;
assert((length % num_dev) == 0);
#pragma omp parallel sections firstprivate(chunksz,num_dev)
{
  for (int dev = 0; dev < NUM_DEVICES; dev++) {
#pragma omp task firstprivate(dev)
    {
      int lb = dev * chunksz;
      int ub = (dev+1) * chunksz;
#pragma omp target device(dev) map(in:y[lb:chunksz]) map(out:x[lb:chunksz])
      {
#pragma omp parallel for
        for (int i = lb; i < ub; i++) {
          x[i] = a * y[i];
        }
      }
    }
  }
}
```



Target update Construct Example

```
#pragma omp target data device(0) map(alloc:tmp[:N]) map(to:input[:N)) map(from:res)
  {
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    update_input_array_on_the_host(input);

#pragma omp target update device(0) to(input[:N])

#pragma omp target device(0)
#pragma omp parallel for reduction(+:res)
    for (i=0; i<N; i++)
      res += final_computation(input[i], tmp[i], i)
  }
```



`map` Clause

```
extern void init(float*, float*, int);
extern void output(float*, int);

void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;
   init(v1, v2, N);

   #pragma omp target map(to:v1[0:N],v2[:N]) \\
                      map(from:p[0:N])
   #pragma omp parallel for
   for (i=0; i<N; i++)
     p[i] = v1[i] * v2[i];

   output(p, N);
}
```

- The `target` construct creates a new *device data environment* and explicitly **maps** the array sections v1[0:N], v2[:N] and p[0:N] to the new device data environment.
- The variable N implicitly mapped into the new device data environment from the encountering task's data environment.

Map-types:

- **`alloc:`** allocate storage for corresponding variable
- **`to:`** alloc and assign value of original variable to corresponding variable on entry
- **`from:`** alloc and assign value of corresponding variable to original variable on exit
- **`tofrom:`** default, both to and form