A Uniform Programming Model for Petascale Computing

Barbara Chapman
University of Houston

WPSE 2009, Tsukuba
March 25, 2009

High Performance Computing and Tools Group
http://www.cs.uh.edu/~hpctools

Agenda

- OpenMP 3.0
- Challenges in Scaling OpenMP
- Heterogeneous Systems / Nodes

The OpenMP Shared Memory API

- High-level directive-based multithreaded programming
  - The user makes strategic decisions
  - Compiler figures out details
  - Threads communicate by sharing variables
  - Synchronization to order accesses and prevent data conflicts
  - Structured programming to reduce likelihood of bugs

```
#pragma omp parallel
#pragma omp for schedule(dynamic)
        for (I=0;I<N;I++){
                NEAT_STUFF(I);
        } /* implicit barrier here */
```

The OpenMP ARB

- OpenMP is maintained by the OpenMP Architecture Review Board (the ARB), which
  - Interprets OpenMP
  - Writes new specifications - keeps OpenMP relevant
  - Works to increase the impact of OpenMP
- Members are organizations - not individuals
  - Current members
    - Permanent: AMD, Cray, Fujitsu, HP, IBM, Intel, Microsoft, NEC, PGI, SGI, Sun
    - Auxiliary: ASCI, cOMPunity, EPCC, KSL, NASA, RWTH Aachen

OpenMP 3.0 Introduces Tasks

- Tasks explicitly created and processed
  - Each encountering thread packages a new instance of a task (code and data)
  - Some thread in the team executes the task

```
#pragma omp parallel
{
  #pragma omp single
   {
    p = listhead ;
    while (p) {
      #pragma omp task
            process (p)
      p=next (p) ;
    }
   }
}
```

Synchronization provided by **#pragma omp taskwait**

Nested Parallelism in OpenMP 3.0

- Per-thread internal control variables
  - Allows, for example, calling `omp_set_num_threads()` inside a parallel region.
  - Controls the team sizes for next level of parallelism
  - Different regions may have different defaults
- Library routines to determine depth of nesting, parent IDs, their team sizes etc.

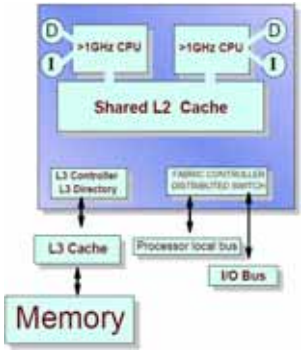
```
omp_get_active_level()
omp_get_ancestor(level)
omp_get_teamsize(level)
```

Version 3.0 ratified by ARB May 2008

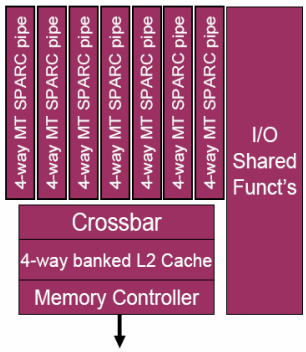
Agenda

- OpenMP 3.0
- Challenges in Scaling OpenMP
- Heterogeneous Systems / Nodes

Multicore Is Everywhere



IBM Power4, 2001

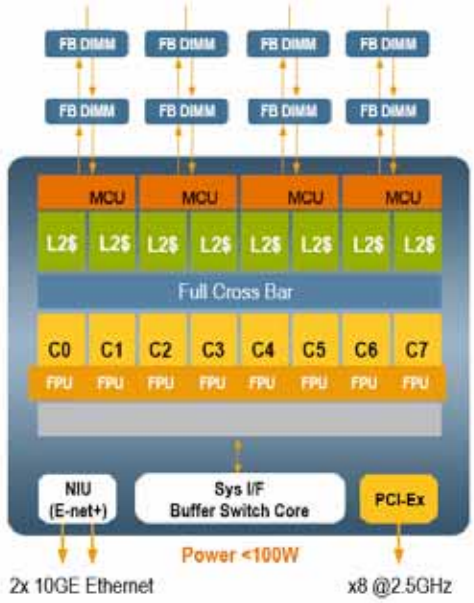


Sun T-1 (Niagara), 2005

Intel rocks the boat 2005

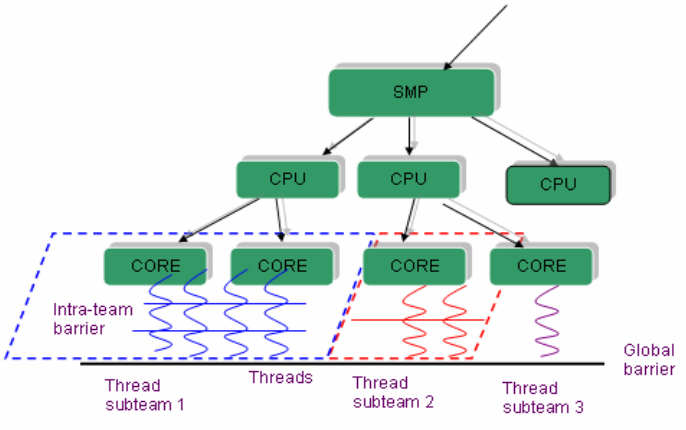
- Small, but growing, number of cores sharing memory
- Individual core may run one or more threads
- Some resources shared between threads (L2 cache, memory bandwidth): details depend on specific architecture
- Introduces need to consider scalability of API and implementation

Niagara 2: True System On a Chip



- 8 cores @1.2GHz - 1.4GHz
- 64 threads per CPU
- Up to 16 FB-DIMMs, 4 memory controllers
  - > Up to 128GB memory (8GB DIMMs)
  - > 2.5x memory BW = 60+GB/S
- 8 x fully pipelined Floating Point units / core
- Dual 10Gbit Ethernet and PCI-E integrated onto chip
- 4MB L2$ (8 banks) 16 way set
- Security co-processor per core
  - > DES, 3DES, AES, RC4, SHA1, SHA256, MD5, RSA to 2048 key, ECC,CRC32

Subteams of Threads?



Thread Subteam: original thread team is divided into several subteams, each of which can work simultaneously. Topologies could also be defined.

- Rather like MPI's groups of pre-existing processes and operations among groups
- Worksharing among groups of pre-existing threads (i.e. a subset of current team of threads)

Increases expressivity of single-level parallelism

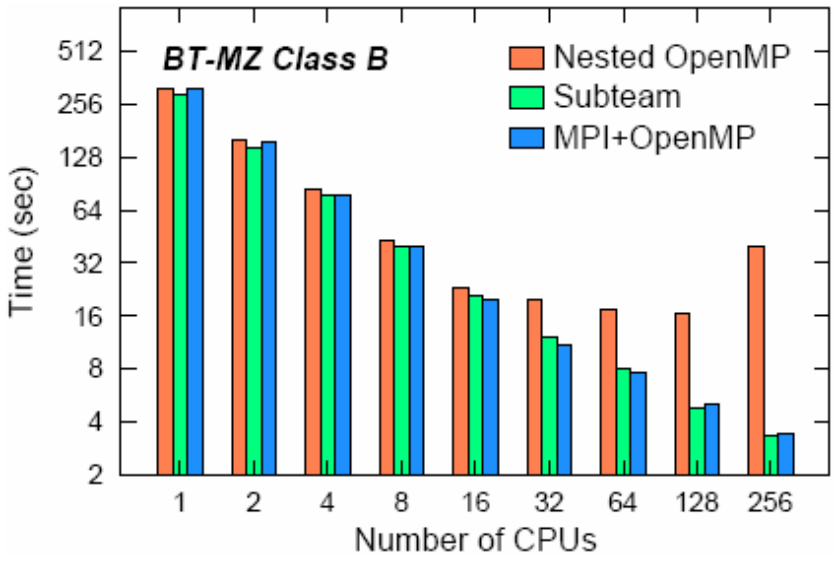
OpenMP Locality: Thread Subteams

```
for (j=0; j<ProcessingNum;j++) {
   #pragma omp for on threads (m:n:k )
    for k=0; k<M;k++) {        //on threads in subteam
         ... Process_data ();
    }                    // barrier involves subteam only
```

- Flexible code region/worksharing/synchronization extension
- Low overhead because of static partition
- Facilitates thread-core mapping for better data locality and less resource contention
- Supports heterogeneity, **hybrid programming**, composition

#pragma omp for on threads (m:n:k)

BT-MZ Performance with Subteams

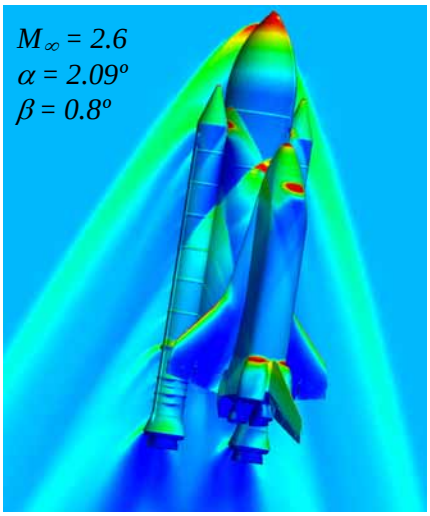


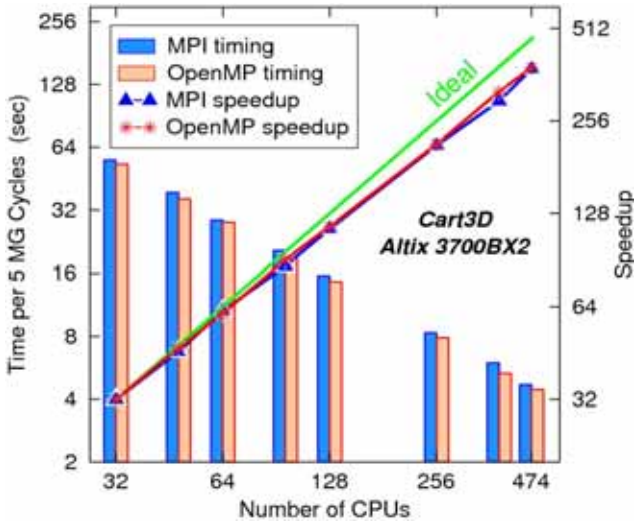
Platform: Columbia@NASA

Subteam: subset of existing team

Cart3D OpenMP Scaling

4.7 M cell mesh Space Shuttle Launch Vehicle example



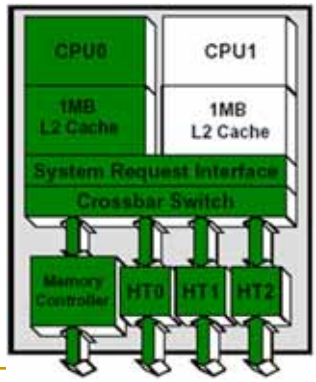


- OpenMP version uses same domain decomposition strategy as MPI for data locality, avoiding false sharing and fine-grained remote data access

OpenMP version slightly outperforms MPI version on SGI Altix 3700BX2, both close to linear scaling.

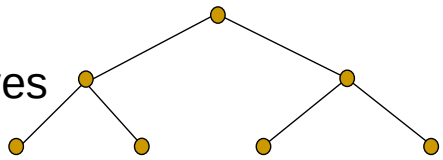
Locality, Locality, Locality

- OpenMP does not permit explicit control over data locality
- Thread fetches data it needs into local cache
- Implicit means of data layout popular on NUMA systems
  - As introduced by SGI for Origin
  - "First touch"
- Emphasis on privatizing data wherever possible, and optimizing code for cache
  - This can work pretty well
  - But small mistakes may be costly



Ideas for Locality Support

- Control thread placement as well as data locality
- Data placement techniques:
  - Rely on implicit first touch or other system support
  - Possibly optimize e.g. via preprocessing step
  - Provide a “next touch” directive that would store data so that it is local to next thread accessing it
- Thread binding techniques:
  - Do this via system calls, command line
  - Programmer hints to “spread out”, “keep close together
  - Logical machine description?
  - Logical thread structure?
- HPF-like data placement directives



“Places” to Enhance Data Locality?

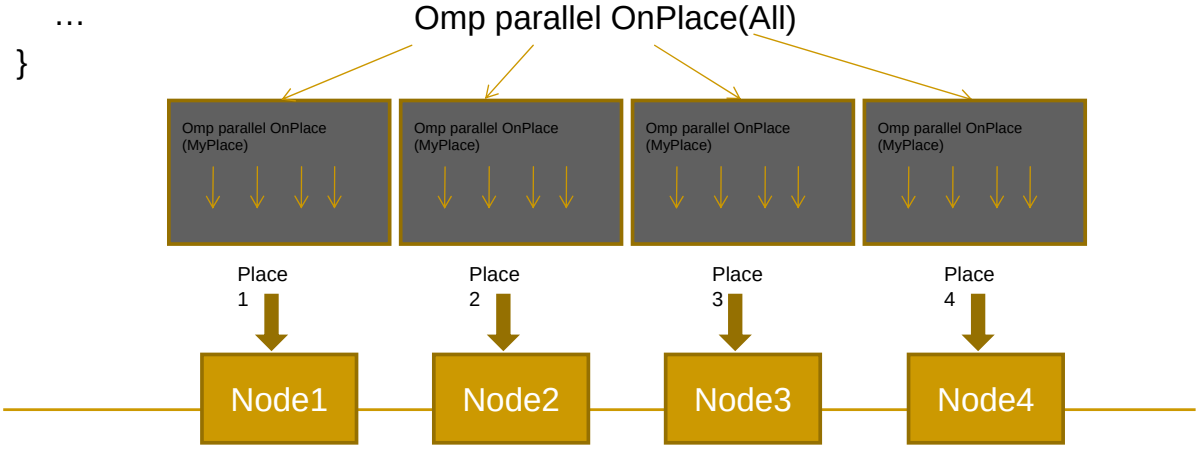
- The place concept is introduced in X10
  - A logical region in the system that data and threads may have affinity with
  - Mapping to hardware nodes at runtime
  - Possible to allocate data within a place
- Could add places to OpenMP
  - Associate worksharing constructs with a place
  - Could permit additional kind of shared memory

#pragma omp task OnPlace(*place*)

Example: Nested Parallelism and Places

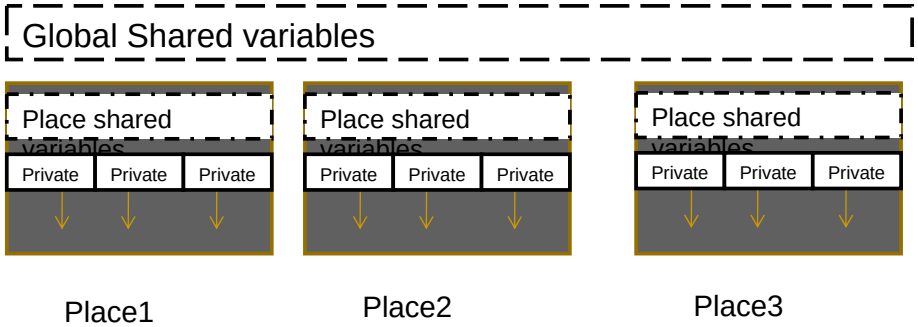
```
#pragma omp place (N) // N is number of places
#pragma omp parallel num_threads(N) OnPlace(All)
{
   int MyPlace = omp_get_place_num();
  #pragma omp parallel OnPlace(MyPlace)
   …
}
```

Point-to-point synchronization might enable interactions between parallel regions



Data Attributes Within a Place

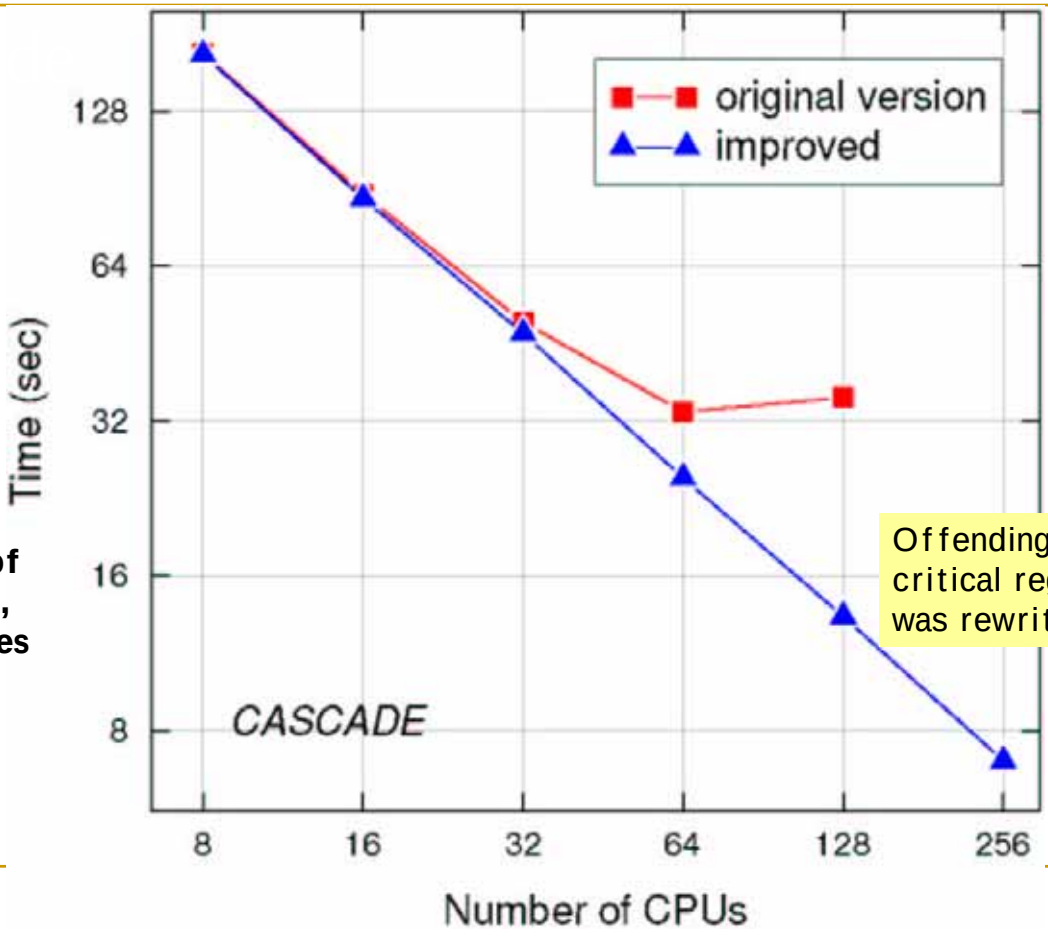
- "Place-shared" variable: a variable shared only within threads in a place



- Default is place-shared if parallel region is associated with a single place

Synchronization Matters

- Reliance on global barriers, critical regions and locks
- Critical region is very expensive
  - High overheads to protect often just a few memory accesses
- It's hard to get locks right
  - And they may also incur performance problems
- Point-to-point synchronization could reduce overall waits
- Condition variables might enable finer-grain synchronization
- Transactions might be an interesting addition
  - Most likely at implementation level only
  - Especially if hardware support provided



Courtesy of R. Morgan, NASA Ames

Offending critical region was rewritten

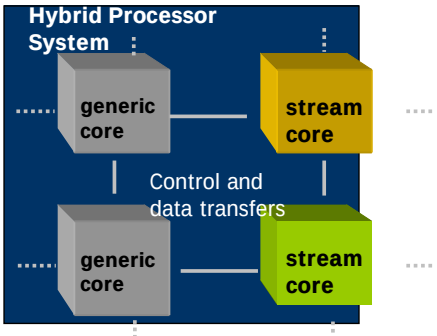
Agenda

- OpenMP 3.0
- Challenges in Scaling OpenMP
- Heterogeneous Systems / Nodes

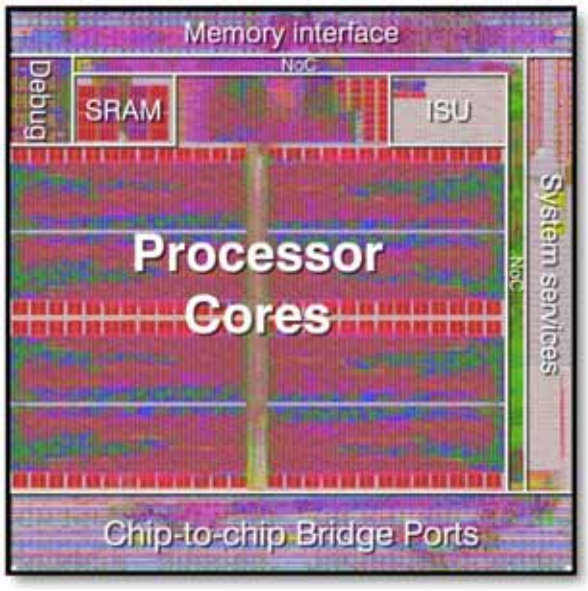
A Heterogeneous World

- Heterogeneous programming is currently very low-level
  - How are we going to program such systems in future?
- If OpenMP is to be used to program a board with devices such as accelerators, GPGUs, extensions are needed

- How to identify code that should be moved to accelerators?
- How to share data between host cores and other devices?
- How is this compiled?
- Debugged?



ClearSpeed Accelerator: CSX600 designed for HPC



- **Processor Core:**
  - 40.32 64-bit GFLOPS
  - 10W typical
  - 210MHz
  - 96 PEs, 6 Kbytes each
  - 8 redundant PEs
- **SoC details:**
  - integrated DDR2 memory controller with ECC support
  - 128 Kbytes of SRAM
- **Design details:**
  - IBM 130nm process
  - 128 million transistors (47% logic, 68% memory)

Streaming

- Create streams for moving data in and out of a special device or a place
  - Program is directed graph of tasks and streams
  - Popular for programming embedded systems
  - Needs to be supported in model for heterogeneous systems
  - But also corresponds to structure of some high-end applications
  - Implementation needs to take care of data motion to/from limited device memories

How to Express Streaming?

```
#pragma omp CreateStreams s1(a), s2(b), s3(c)
#pragma omp task in(s1) out(s2)
   converter(s1, s2);
#pragma omp task in(s2) out(s3)
   compression(s2,s3);
 }
```

Create streams and associate them with data

Link streams with tasks/worksharing to define input and output streams

The pair of tasks will be executed in pipelined fashion

Alternative: have in/out clauses associated with parallel regions. The variables may be passed via point-to-point synchronization constructs.

Example: Heterogeneous Extensions

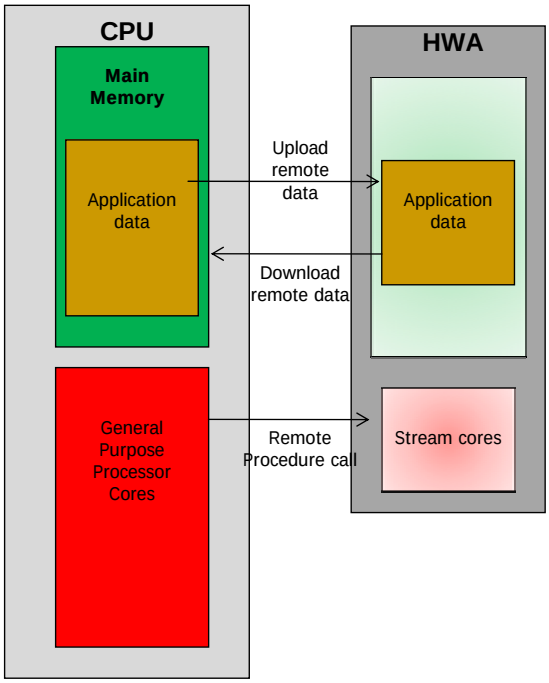
- PGI has introduced OpenMP-like directives
  - To specify regions whose loops will be compiled for acceleration as far as possible
  - User may specify device, input and output data, portions for sequential execution, unrolling, SIMD..
  - Compiler attempts to translate for target device

```
#pragma omp parallel for SIMD(32) // inner level parallelism
  for (int j=0; j<N; j++)
    a[i,j] = b[i][j] * c[i][j]
 }
```

Note: PGI does not use OMP pragmas

Example: CAPS HMPP

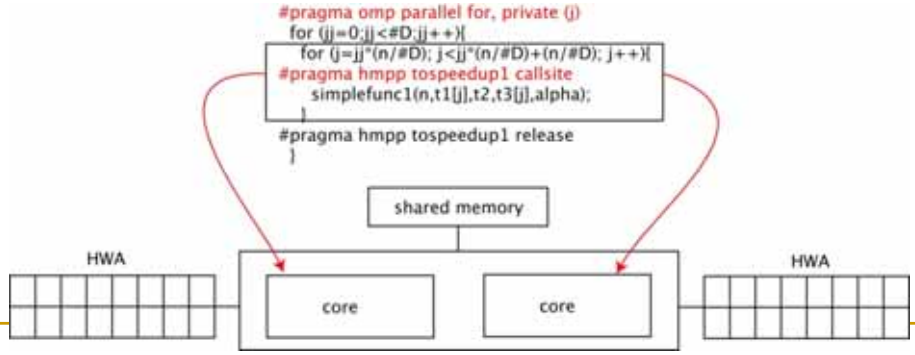
- Declare hardware specific implementations of functions (HMPP codelets)
  - Can be specialized to the execution context (data size, ...)
- Codelet calls (RPC)
  - Synchronous, asynchronous properties
- Data transfers
  - Data prefetching
- Synchronization barriers
  - Host CPU will wait until remote computation is complete



CAPS: Multiple Devices

- Use #D accelerators in parallel

```
#pragma omp parallel for, private (j)
  for (jj=0;jj<#D;jj++){
    for (j=jj*(n/#D); j<jj*(n/#D)+(n/#D); j++){
#pragma hmpp tospeedup1 callsite
      simplefunc1(n,t1[j],t2,t3[j],alpha);
    }
#pragma hmpp tospeedup1 release
  }
```



Heterogeneous Large-Scale Systems?

- Parallel region across machine, needs way to specify mapping of shared data at this level
- Inner level of parallel regions, mapped to places by application developer
  - Shared data is at same place
- In/out clauses to specify data that may be transferred between regions
- Additional levels of parallel regions to map code to accelerators, also with in/out clauses

Summary

- OpenMP needs extensions if it is to be a useful high-end programming model
- Locality support is essential
- Heterogeneity is present in high-end, general-purpose and embedded systems
- To support heterogeneity, OpenMP also needs some extensions
- Placement of code, more data locality support
- Compiler technology needs to be developed