Directive-Based Parallel Programming at Scale?

Barbara Chapman
Stony Brook University
Brookhaven National Laboratory

Charm++ Workshop, April 19 2016

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

HPC TOOLS

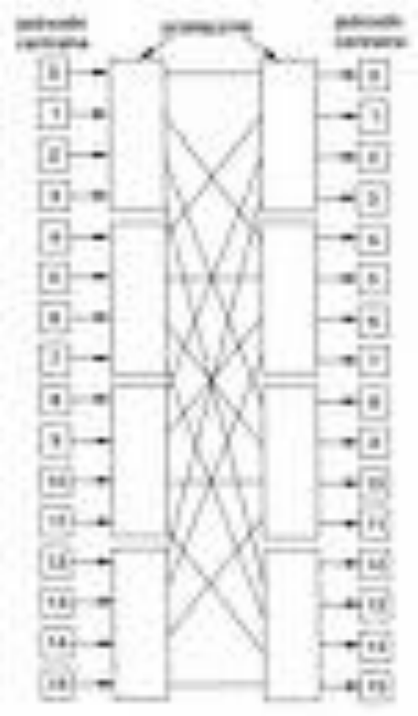
http://www.cs.uh.edu/~hpctools

Agenda

- Directives: A little (pre)history
- Evolving the standard
- Today's challenges
- Where to next?

Symmetric Multiprocessors

- 1980s saw attempts to build parallel computers with shared memory
  - Alliant
  - Sequent
  - Encore, …
- Programmed using Fortran
  - Vendor extensions, mainly to parallelize loops
- Attempt to develop standard API
  - PCF features for loop parallelism in Fortran code
  - Fortran standards subcommittee formed



BBN Butterfly

- Every CPU able to access memory associated with other CPUs
- Big penalty for non-local access
- 15 times slower than local memory access

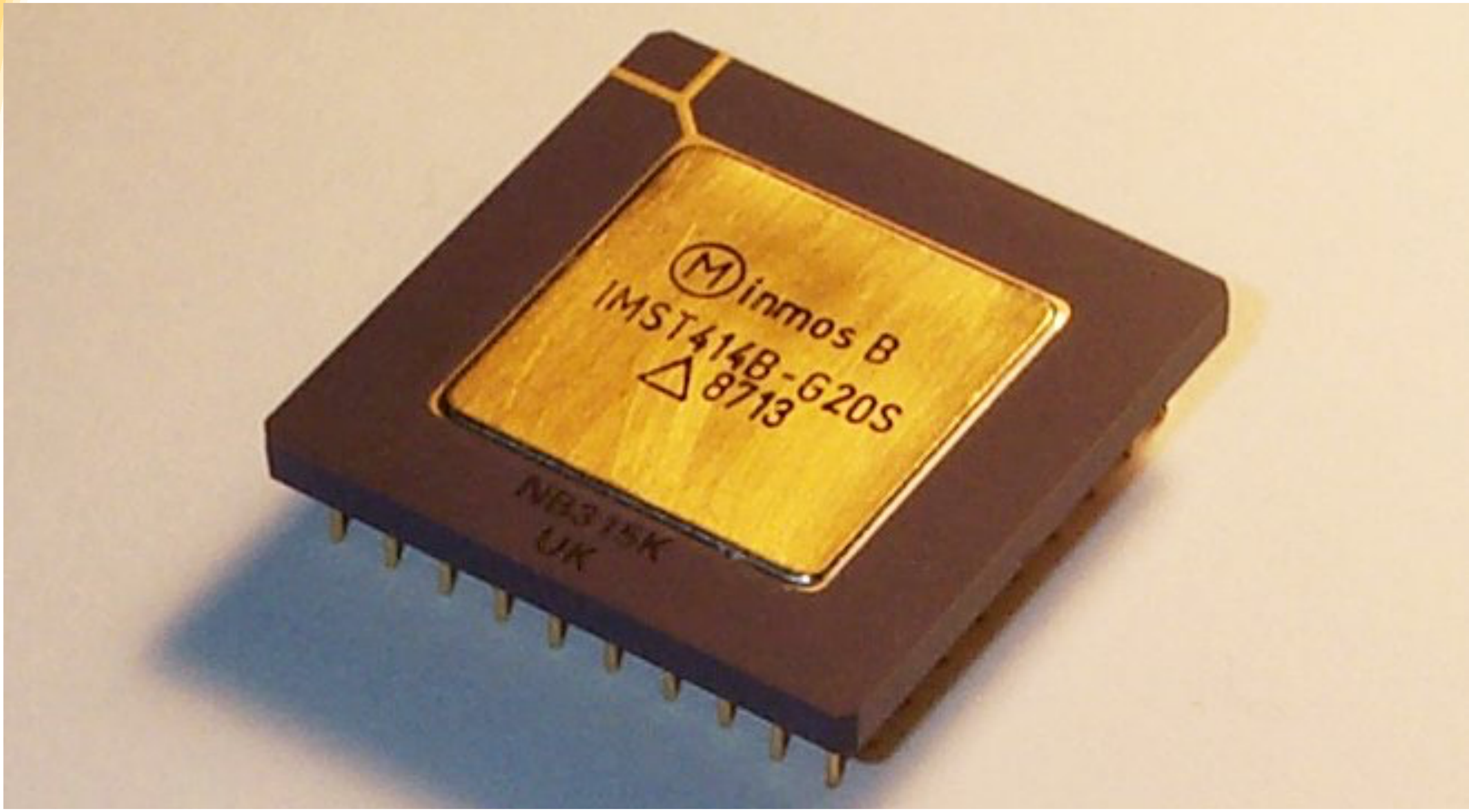
PCF Example

```
PARALLEL SECTIONS
SECTION
PARALLEL DO  I= i , N
    A(I) = B(I) * C(I)
END DO
SECTION
PARALLEL DO J = i, M
    D(J) = F(J) / E(J)
END DO
END PARALLEL SECTIONS
```

- Proposed as Fortran extensions
- Team of threads execute parallel construct
- Parallel loops
- Parallel sections
- Critical, locks and post / wait
- Ordered execution

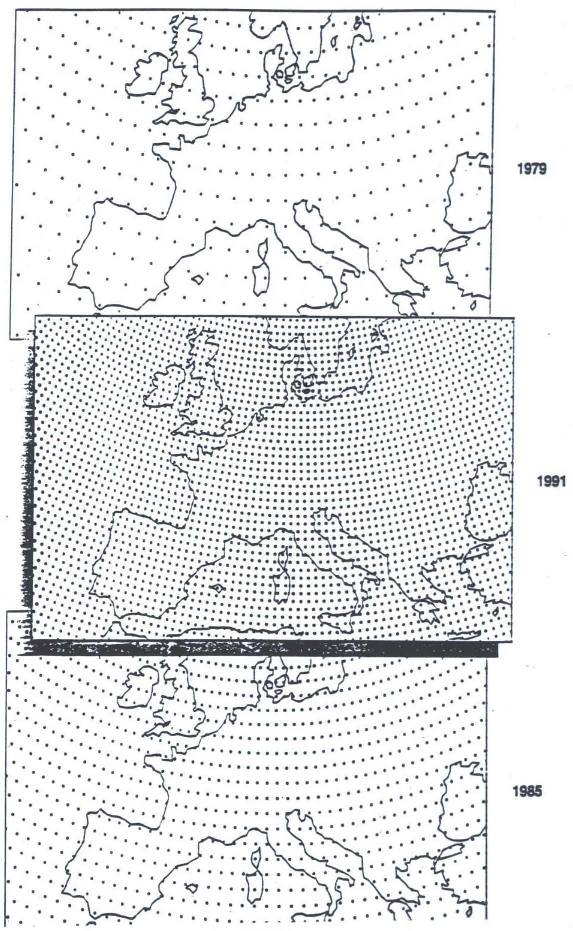
- Loop iterations distributed among threads by implementation; must be iteration-order independent
- Sections of code must be data independent
- Shared and private variables
- Loop variable undefined outside parallel loop construct
- Nested parallelism

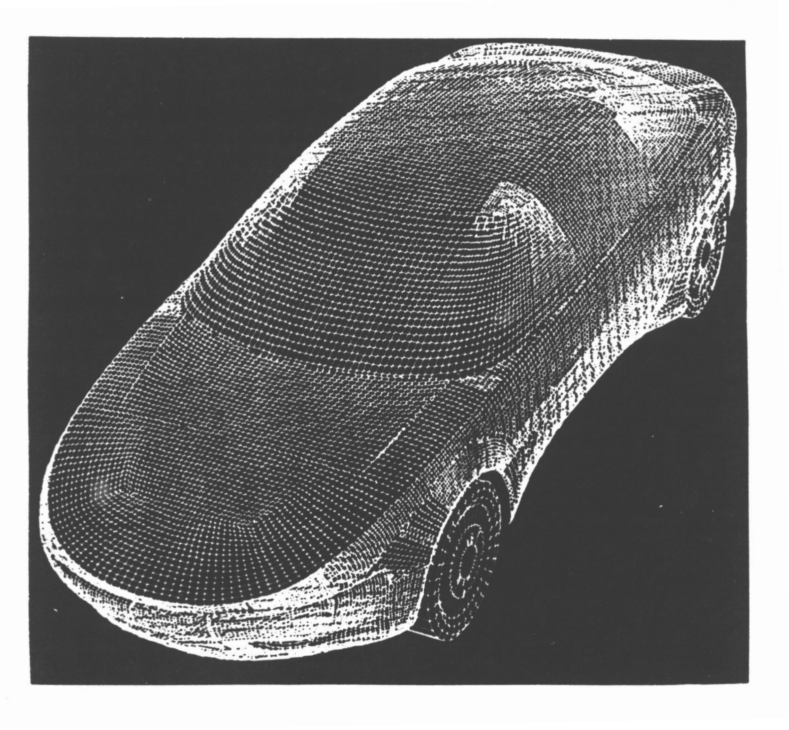
A New Kind of Architecture, Late ‘80s



CM-5 TOP500 #1 June 1993

Using The Compute Power





High Performance Fortran (HPF)

- Directives extend Fortran for distributed memory parallel programming
  - First definition early 1993, revision 1997
  - Japanese created additional features in JA-HPF
- Main features are **directives** for data mapping and parallel loops
  - Work performed where the data is stored
  - Some library routines
- Broad participation in standards effort

HPF Example

Parallel Do
Forall
Independent

```
!HPF$  DISTRIBUTE W ( BLOCK )
!HPF$  INDEPENDENT, NEW ( X ),  REDUCTION ( SUM )

          DO I = 1, N
             X = W(I) * (I - 0.5)
             SUM = SUM + F ( X )
          END DO
```

* Team of processes execute entire program
* Loop iterations are distributed among processes based on data distribution
* Communication at end of loop to obtain global value SUM

* Each process has local segment of W
* Each process has its own copy of variable X
* Each process computes local value of SUM
* SUM updated at end of loop, result replicated

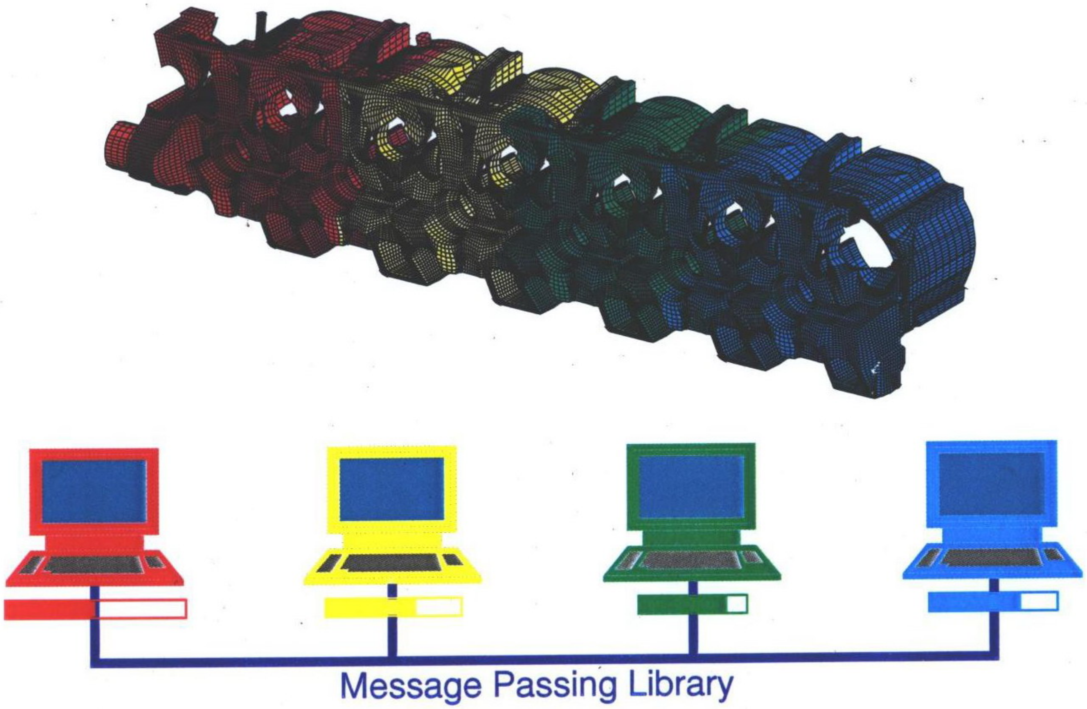
What Happened to HPF?

- Compilers slow to arrive, and supported different styles of HPF programming
  - Based upon Fortran 90, also slow to mature
- Considered suitable for structured (regular) grids only
- MPI flexible and established by the time HPF compilers matured
  - Codified experience with early comms libraries
- Japanese vendors continued to add features and provide compilers after others gave up

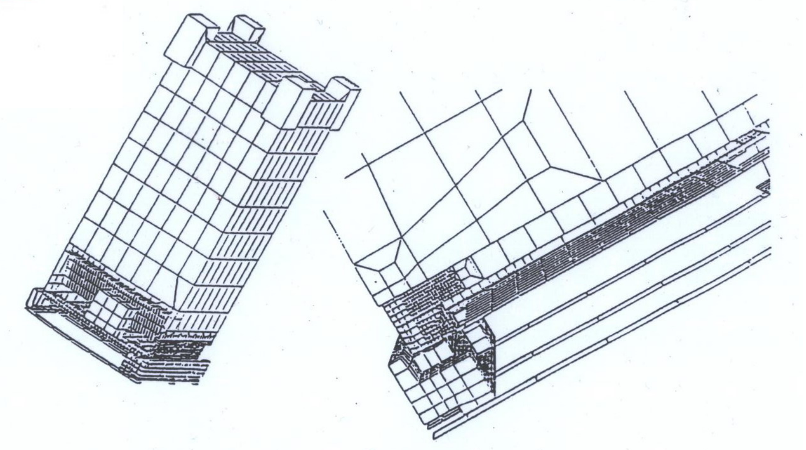
HPF User Experience

- HPF application development was hard
  - Required global modifications
  - incremental development not possible
- **Users had little insight into execution behavior**
  - Creation of good HPF code required insight into compilation process
  - But this was rare
  - Performance degradation could be severe
- Benefits of directive approach neither experienced nor understood by many
- Not surprisingly, **few tools** available (HPF version of Totalview was created)

MPI Becomes Widely Used



User distributes the data and computation explicitly to system processing nodes.



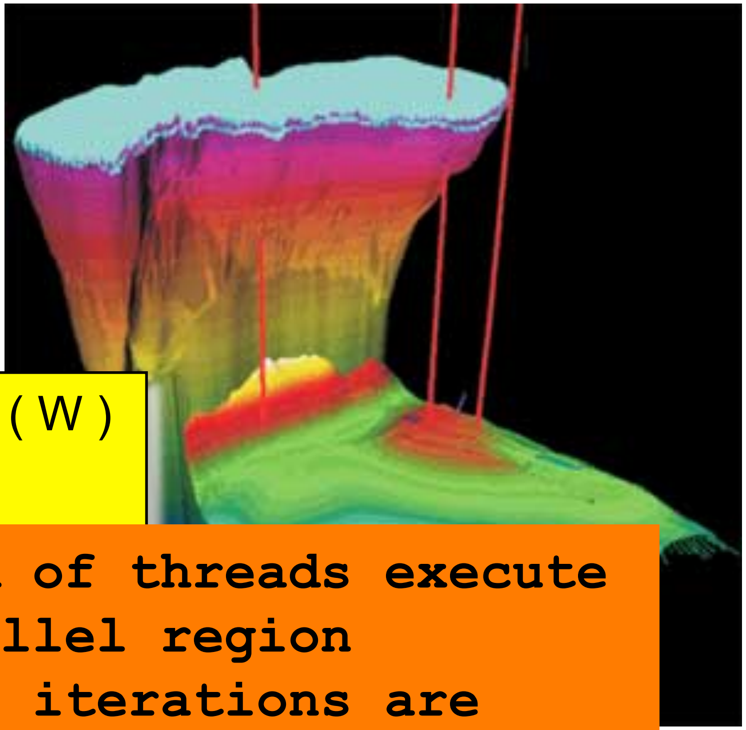
Example simulation for a packaged Refrigerator

Return of Shared Memory

Sun Fire 6800 Server, 24 CPUs

- SMPs on desktop, late 1990s (HP, Sun, Intel, IBM, …)
  - Mainstream market, general-purpose applications
  - Mostly 2 – 4 cache coherent CPUs
  - A few bigger systems e.g. Sun's 6400 (144 CPUS)
- Large-scale distributed shared memory (DSMs)
- Memory is distributed, but globally addressed
  - E.g. HP Exemplar, SGI Origin and Altix series
  - Looks like shared memory system to user
  - Hardware supports cache coherency
  - Origin: non-local data twice as slow

OpenMP Example



```
!$OMP PARALLEL DO PRIVATE ( X ) , SHARED ( W )
!$OMP& REDUCTION ( +: SUM )
            DO I = 1, N
                X = W(I) * (I - 0.5)
                SUM = SUM + F ( X )
            END DO
!$OMP END PARALLEL
```

* Team of threads execute parallel region
* Loop iterations are distributed among threads
* Implicit synchronization at end of region

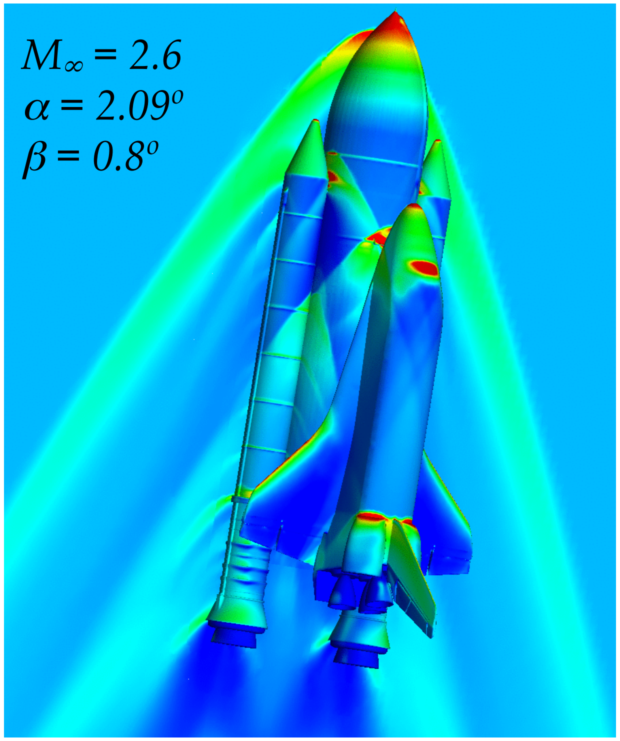
* All threads access same W
* Each executing thread has its own copy of variable X
* Each thread creates and initializes a private copy of shared variable SUM.
* SUM is updated at next synchronization point

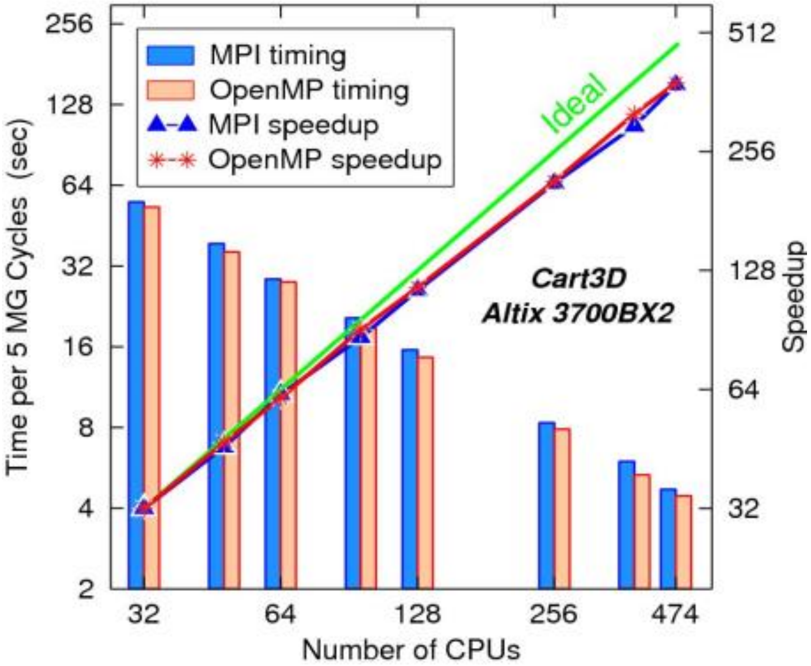
Agenda

- Directives: A little (pre)history
- Evolving the standard
- Today's challenges
- Where to next?

Cart3D OpenMP Scaling, ca. 2005

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.

Data Mapping and Affinity

Proposed OpenMP Extensions, 1999

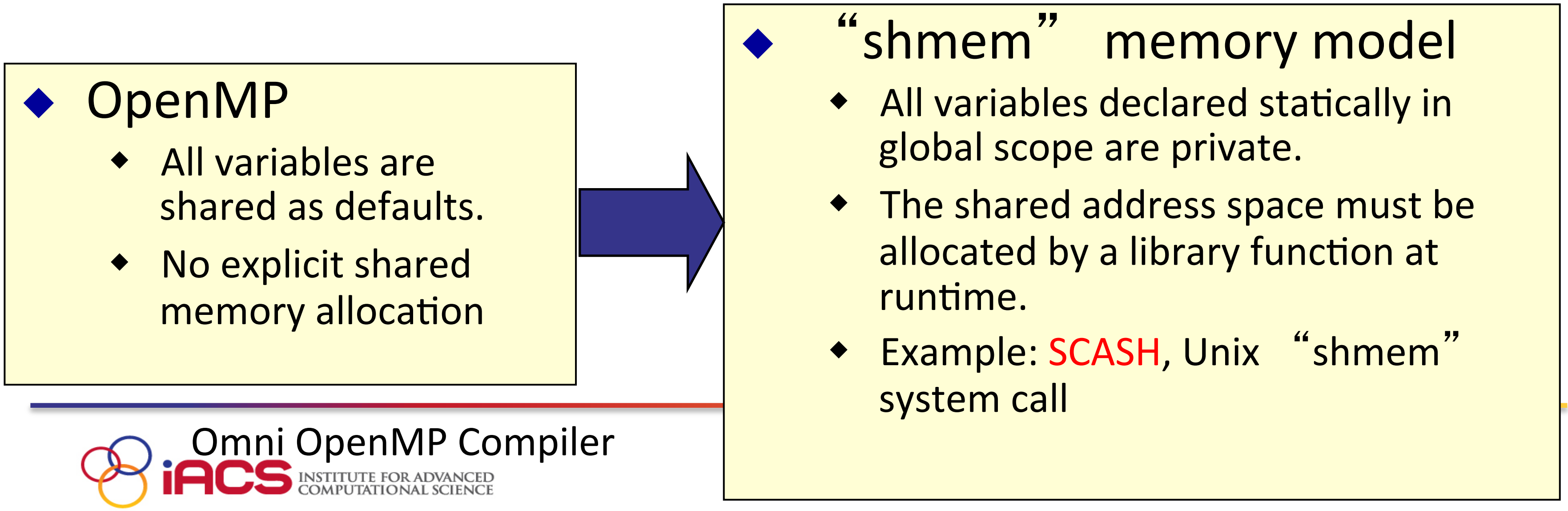
- SGI page-based data distribution extensions
  - Allocates *pages* to memory across system nodes
  - Preserves illusion of true shared memory
- HPF-style data mappings
  - Didn't do well on page-based system
  - SGI, Compaq

"first-touch" default mapping works pretty well (if developer is aware of it)

```
!$SGI     DISTRIBUTE array ( CYCLIC (1) )
!$OMP     PARALLEL DO PRIVATE ( i , active)
!$OMP&    SHARED (  level )
!$SGI+    AFFINITY (i) = DATA ( array ( i ) )
          DO i = 1,  max
             IF (   array ( i )  >= 1) then
                active = ....
                CALL solve ( active, level, …)
             END IF
          END DO
```

Omni Compiler: Cluster-enabled OpenMP, 2002

- OpenMP for a cluster (distributed memory system)
  - message passing library (MPI, PVM) provides high performance, but difficult and cumbersome.
- Use software distributed shared memory system SCASH as underlying runtime system on cluster
  - Page-based DSM
  - Related Work: OpenMP compiler for TreadMarks by Rice (later clOMP)



- OpenMP
  - All variables are shared as defaults.
  - No explicit shared memory allocation

- “shmem” memory model
  - All variables declared statically in global scope are private.
  - The shared address space must be allocated by a library function at runtime.
  - Example: SCASH, Unix “shmem” system call

OpenMP 3.0 Introduces Tasks, 2008

- Tasks explicitly created and processed

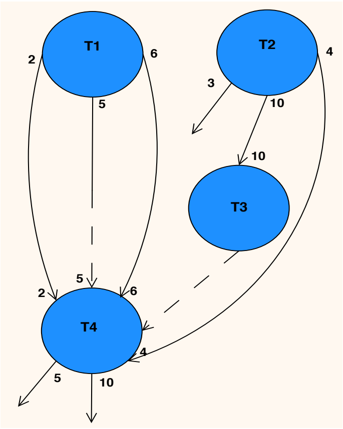
  - Each encountering thread packages a new instance of a task (code and data)
  - Some thread in the team executes the task

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

Asynchronous Task Dependence

- Increase power of tasks, reduce barrier synchronization
- **Task synchronization constructs**
  - **taskwait,** and **barrier** construct

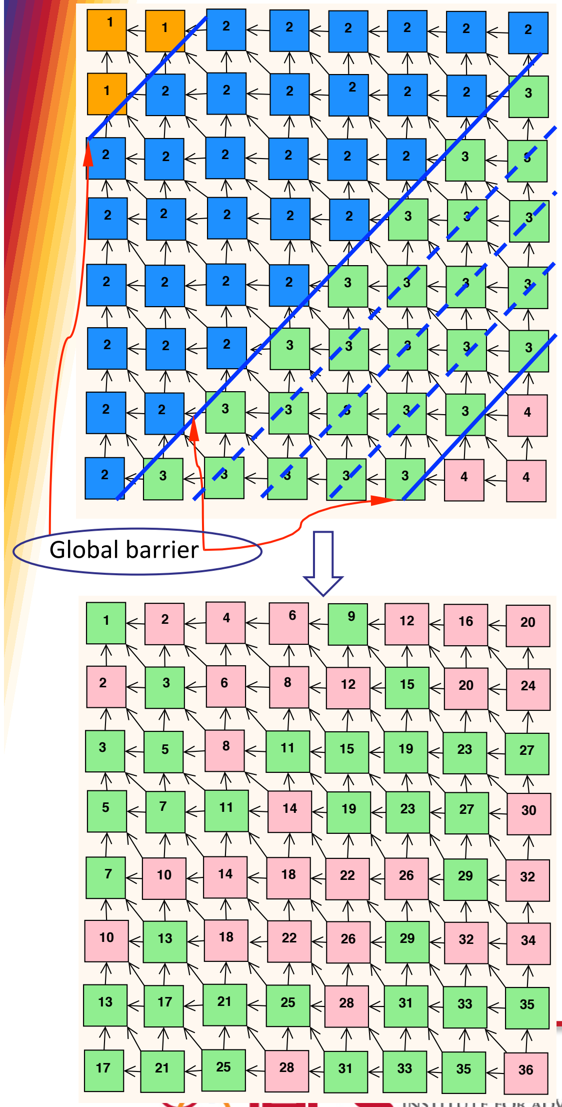
```
int fib(int n) {
   int x, y;
   if (n < 2)  return n;
   else {
        #pragma omp task shared(x)
        x = fib(n-1);
        #pragma omp task shared(y)
        y = fib(n-2);
        #pragma omp taskwait
        return x + y;
   }
}
```

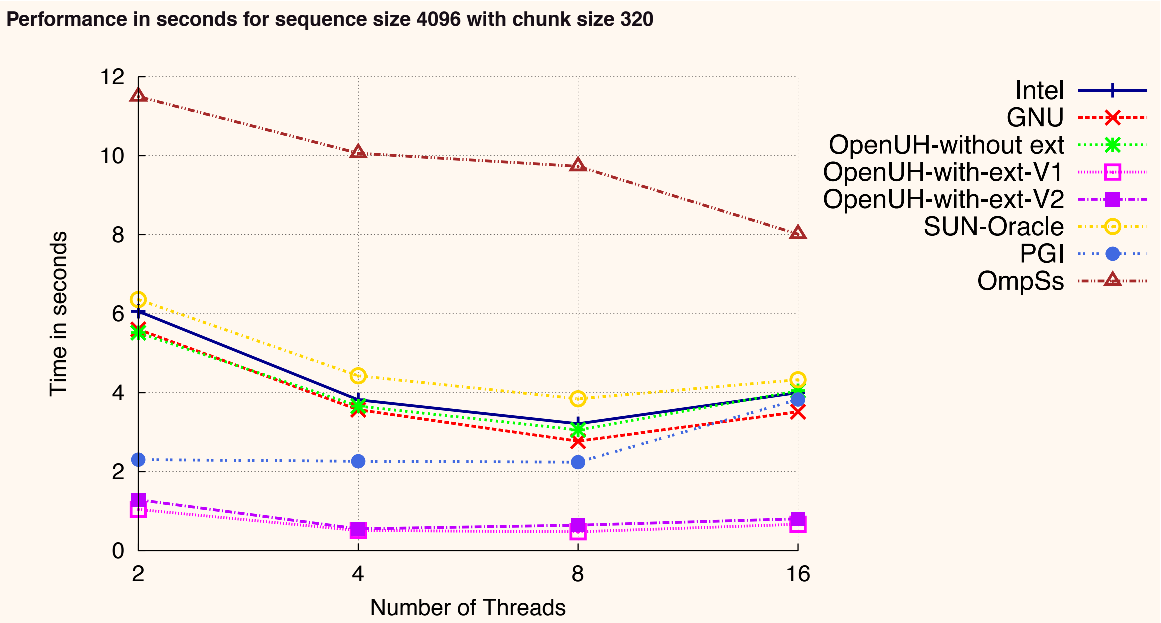


#pragma omp task depend (out: t1, t2, …) depend (in: t4, t5)

- Avoid the use of global locks
- Work with workstealing
- Decentralized dependency setup and resolution

Eliminating Global Barriers in Smith-Waterman

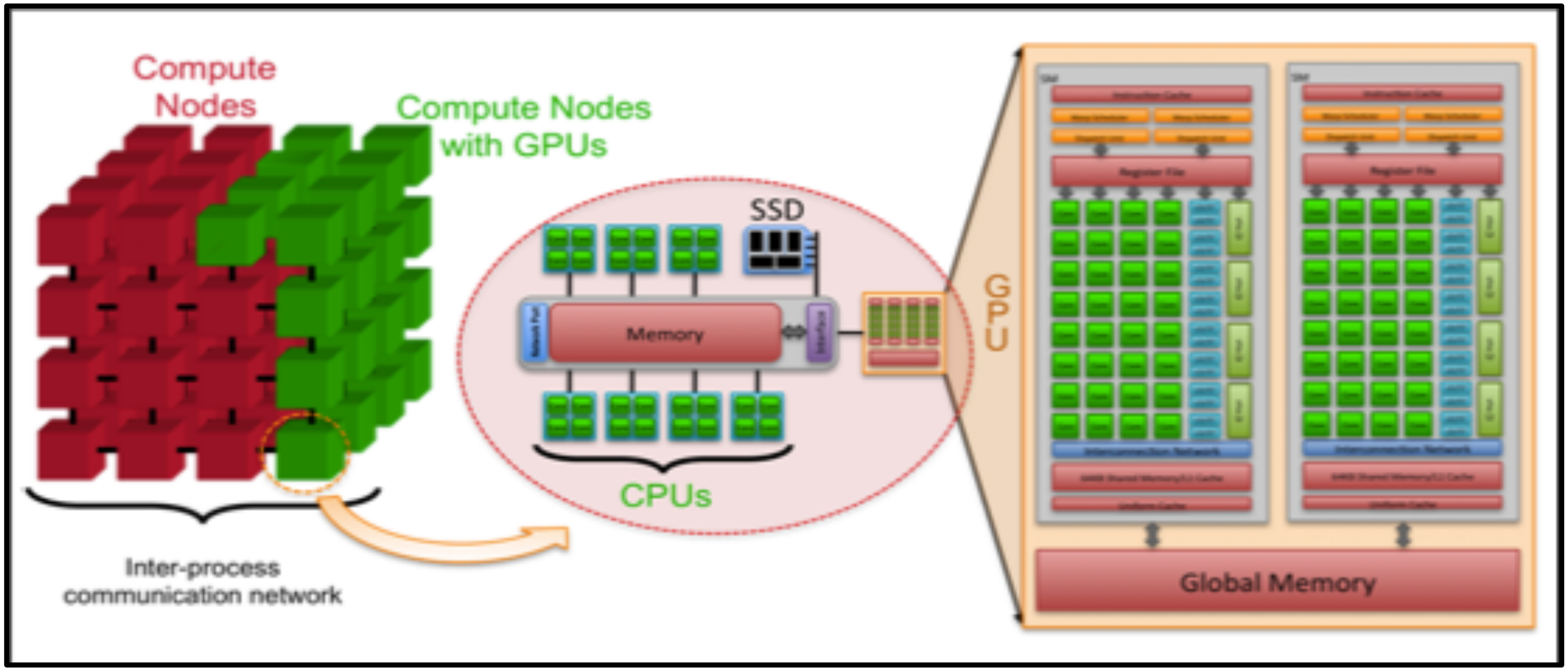




Threads	OpenUH_ext	OmpSs	Quark
2	1.045	52.251	2.639
4	0.511	50.640	2.278
8	0.480	48.645	2.081
16	0.669	46.256	2.395

A Prototype Implementation of OpenMP Task Dependency Support; Priyanka Ghosh, Yonghong Yan, Deepak Eachempati and Barbara Chapman; International Workshop on OpenMP (IWOMP) 2013

Core Heterogeneity in HPC Systems

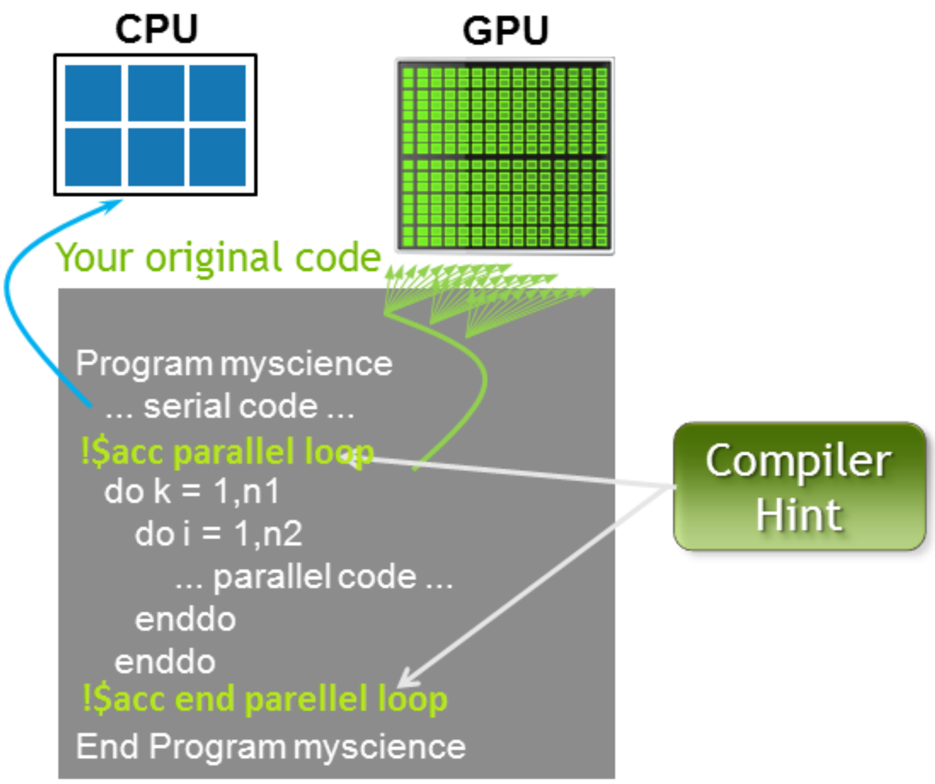


Each node has multiple CPU cores, and some of the nodes are equipped with additional computational accelerators, such as GPUs.

www.olcf.ornl.gov/wp-content/uploads/.../Exascale-ASCR-Analysis.pdf

OpenACC

- Directive-based programming for offloading code to accelerators
  - For Fortran, C, C++
  - Loop-based computations
- Compute directives
  - *parallel*: control to the user
  - *kernels*: freedom to the compiler
- Three levels of parallelism: gang, worker and vector
- Open-source and proprietary implementations
- OpenACC Validation Suite
  - C and Fortran validation for OpenACC 2.0
- SPEC Accelerator Benchmarks



```
Program myscience
  ... serial code ...
!$acc parallel loop
  do k = 1,n1
    do i = 1,n2
        ... parallel code ...
    enddo
  enddo
!$acc end parellel loop
End Program myscience
```

http://www.openacc-standard.org/

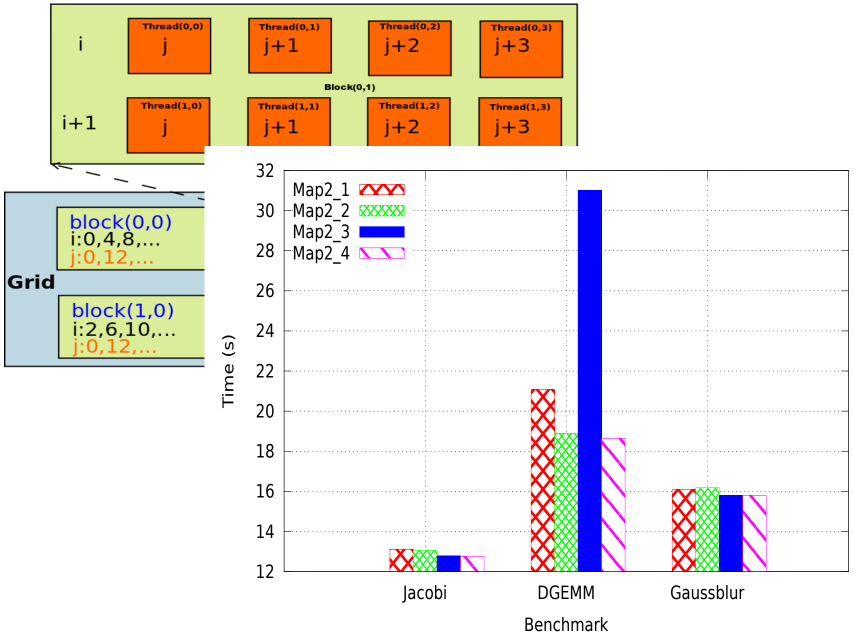
Agenda

- Directives: A little (pre)history
- Evolving the standard
- Today's challenges
- Where to next?

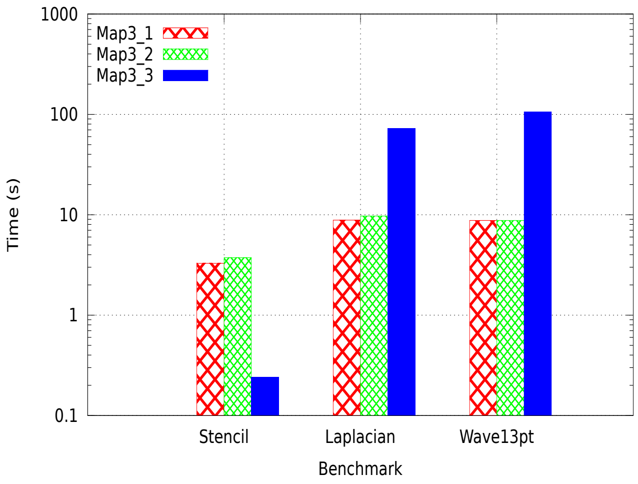
OpenACC Compiler Translation

- Need to achieve coalesced memory access on GPUs

```
#pragma acc loop gang(2) vector(2)
for ( i = x1; i < X1; i++ ) {
#pragma acc loop  gang(3) vector(4)
for ( j = y1; j < Y1; j++ ) {...... }
}
```



Double nested loop mapping.



Triple nested loop mapping.

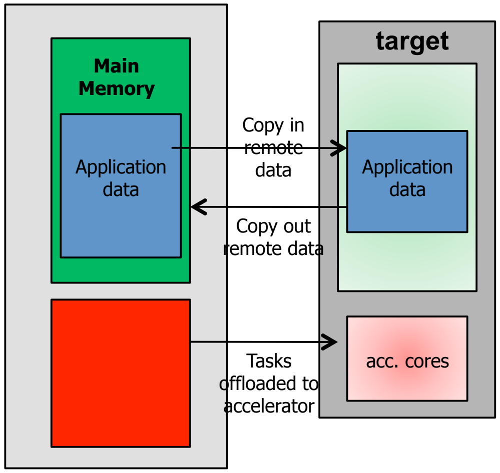
Compiling a High-level Directive-Based Programming Model for GPGPUs; Xiaonan Tian, Rengan Xu, Yonghong Yan, Zhifeng Yun, Sunita Chandrasekaran, and Barbara Chapman; 26th International Workshop on Languages and Compilers for Parallel Computing (LCPC2013)

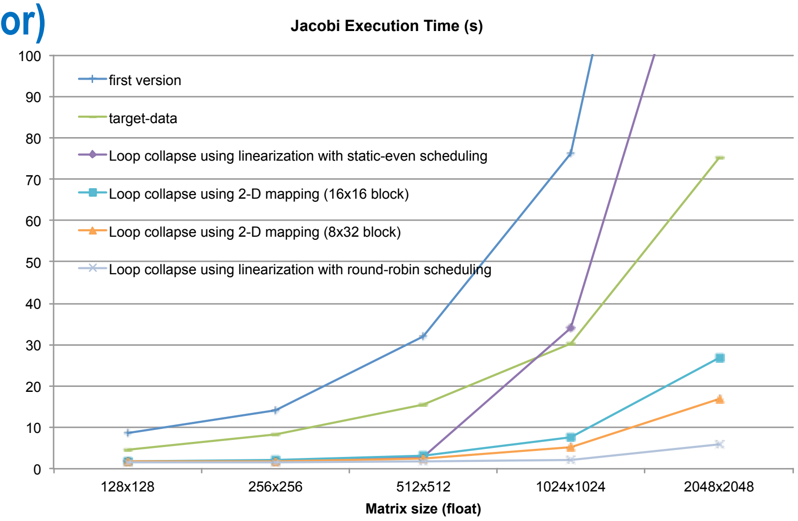
OpenMP for Accelerators

```
#pragma omp target data device (gpu0) map(to:n, m, omega, ax, ay, b, \
  f[0:n][0:m])  map(tofrom:u[0:n][0:m]) map(alloc:uold[0:n][0:m])

while ((k<=mits)&&(error>tol))
{
// a loop copying u[][] to uold[][] is omitted here
 …
   #pragma omp target  device(gpu0)

#pragma omp parallel for private(resid,j,i) reduction(+:error)
for (i=1;i<(n-1);i++)
  for (j=1;j<(m-1);j++)
  {
    resid = (ax*(uold[i-1][j] + uold[i+1][j])\
        + ay*(uold[i][j-1] + uold[i][j+1])+ b * uold[i][j] - f[i][j])/b;
    u[i][j] = uold[i][j] - omega * resid;
    error = error + resid*resid ;
  } // rest of the code omitted  ...
}
```

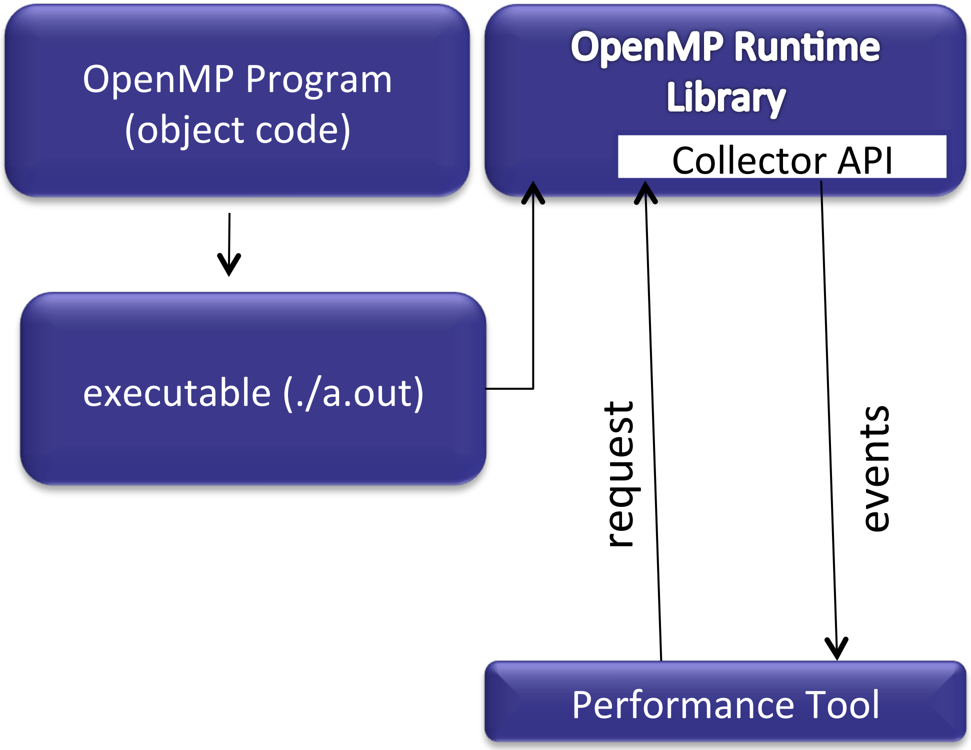




Early Experiences With The OpenMP Accelerator Model; Chunhua Liao, Yonghong Yan, Bronis R. de Supinski, Daniel J. Quinlan and Barbara Chapman; International Workshop on OpenMP (IWOMP) 2013, September 2013

Dynamic Program Adaptation

- OpenMP fairly amenable to dynamic adaptation
  - Adjustment of thread count, schedule
  - Adaptive barriers, reduction routines
  - Runtime decisions
  - Tasks, mergeable
- Use of performance interface to inform dynamic tools
  - Can help adjust data layout, find memory performance problems
- Potential useful for variety of runtime techniques



False Sharing: Monitoring Results

- Cache line invalidation measurements

Program name	1-thread	2-threads	4-threads	8-threads
histogram	13	**7,820,000**	**16,532,800**	**5,959,190**
kmeans	383	28,590	47,541	54,345
linear_regression	9	**417,225,000**	**254,442,000**	**154,970,000**
matrix_multiply	31,139	31,152	84,227	101,094
pca	44,517	46,757	80,373	122,288
reverse_index	4,284	89,466	217,884	**590,013**
string_match	82	**82,503,000**	**73,178,800**	**221,882,000**
word_count	4,877	**6,531,793**	**18,071,086**	**68,801,742**

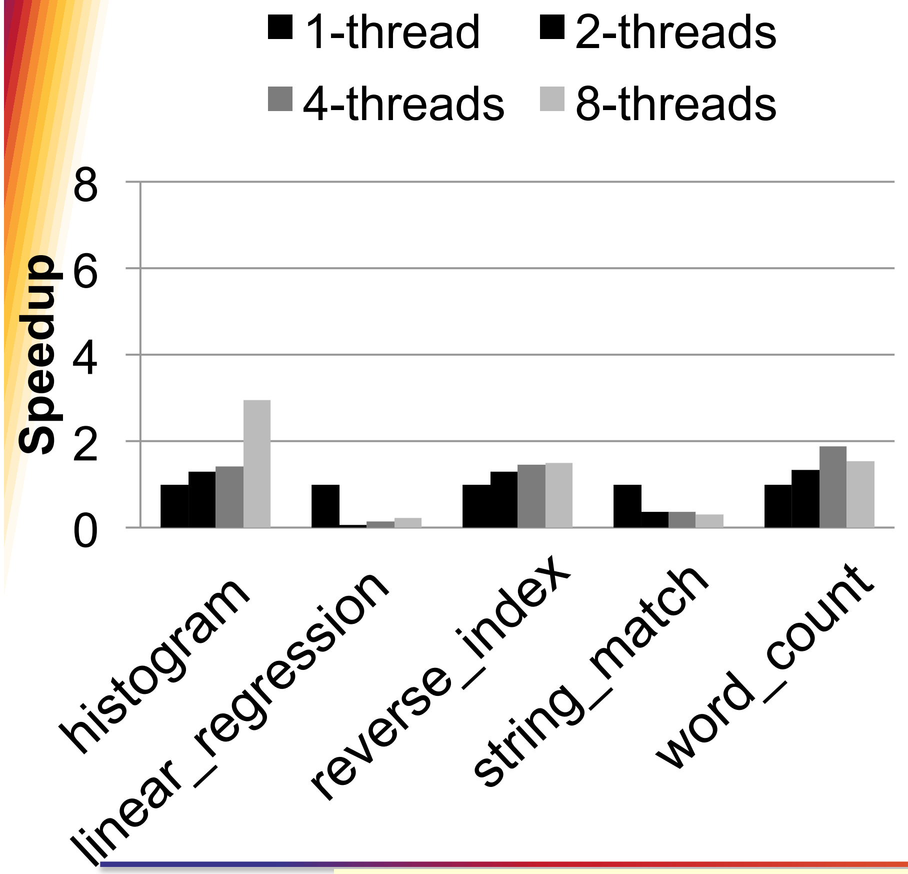
False Sharing: Data Analysis Results

- Determining the variables that cause misses

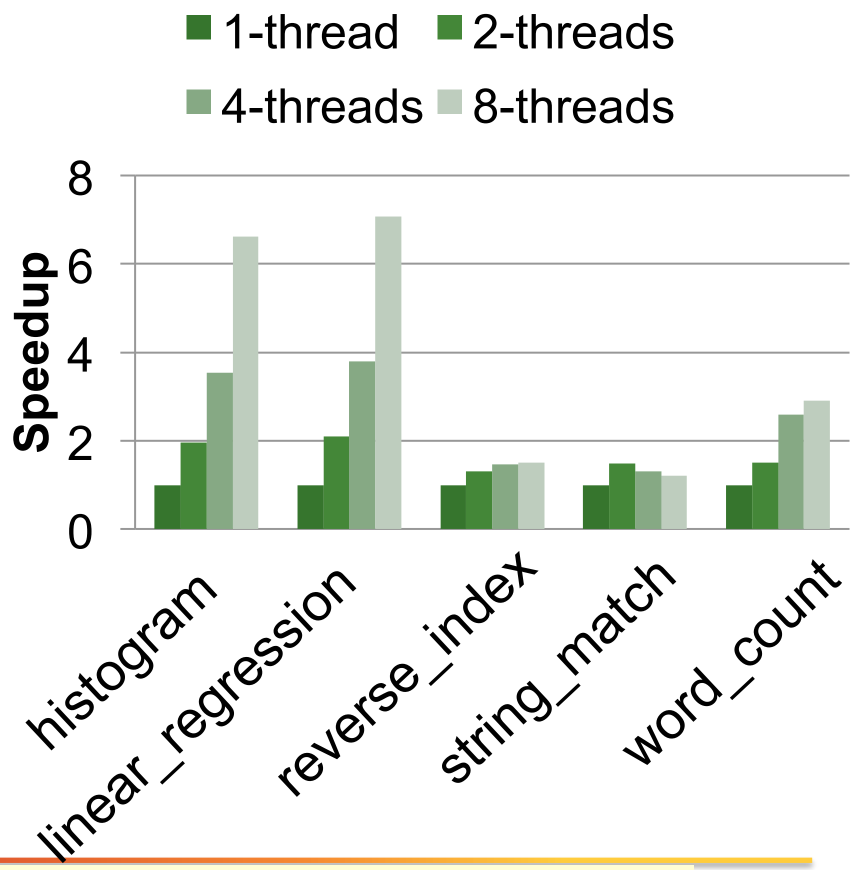
Program Name	Global/static data	Dynamic data
histogram	-	main_221
linear_regression	-	main_155
reverse_index	use_len	main_519
string_match	key2_final	string_match_map_266
word_count	length, use_len, words	-

Runtime False Sharing Detection

Original Version



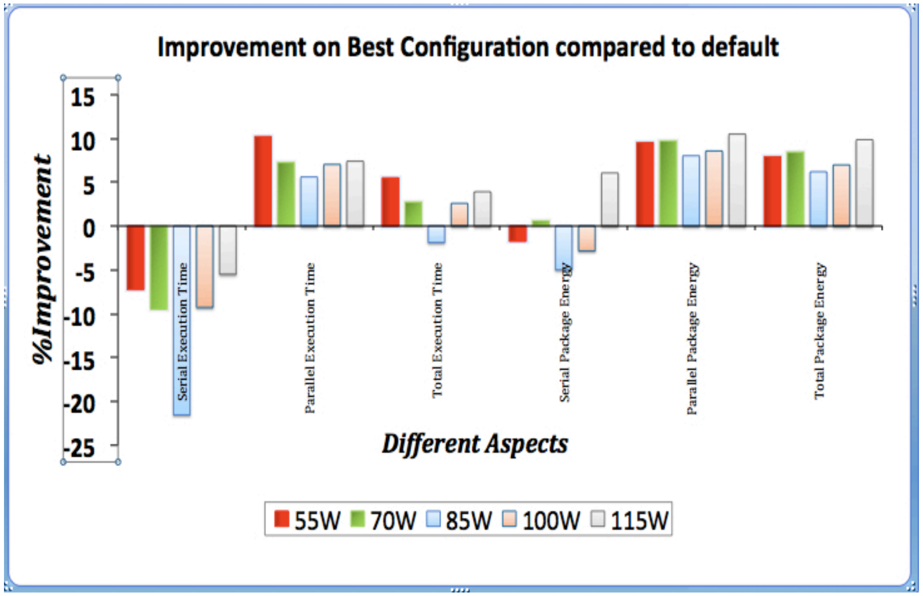
Optimized Version



B. Wicaksono, M. Tolubaeva and B. Chapman. "Detecting false sharing in OpenMP applications using the DARWIN framework", LCPC 2011

Energy Management Tools

- OpenMP runtime settings can be adjusted statically and dynamically for best performance
  - Number of threads, scheduling policy and chunk size, wait policy, binding policy, may all affect performance
- Selections are not independent of power cap
- Modeling may help select settings to optimize both energy and execution performance

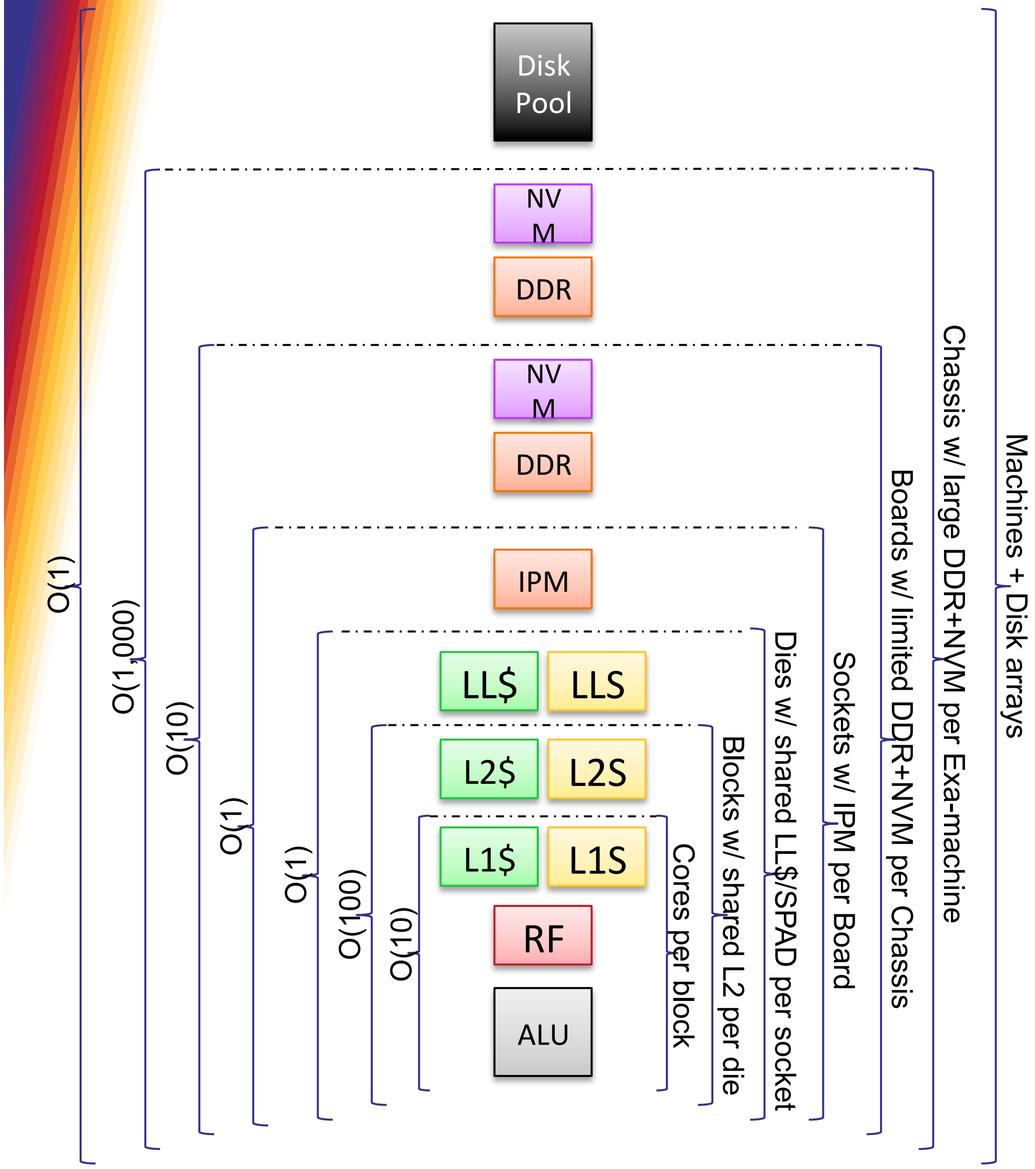


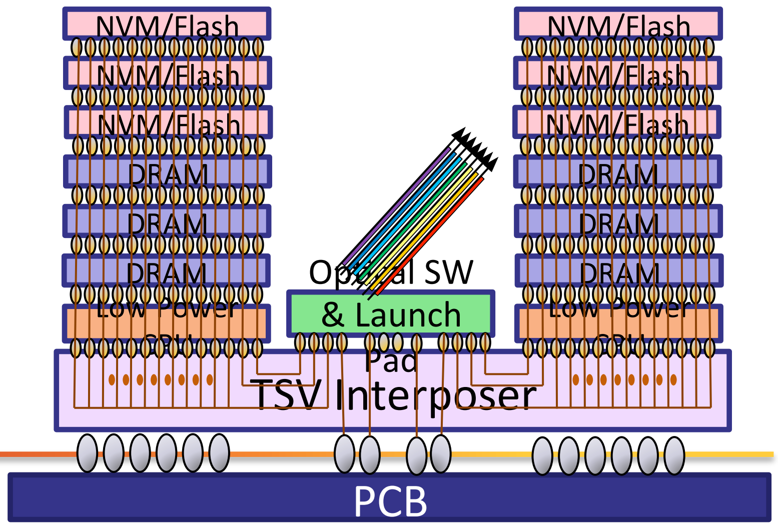
%-age improvement in Co-MD application under different power capping

Agenda

- Directives: A little (pre)history
- Evolving the standard
- Today's challenges
- Where to next?

Memory Will Change





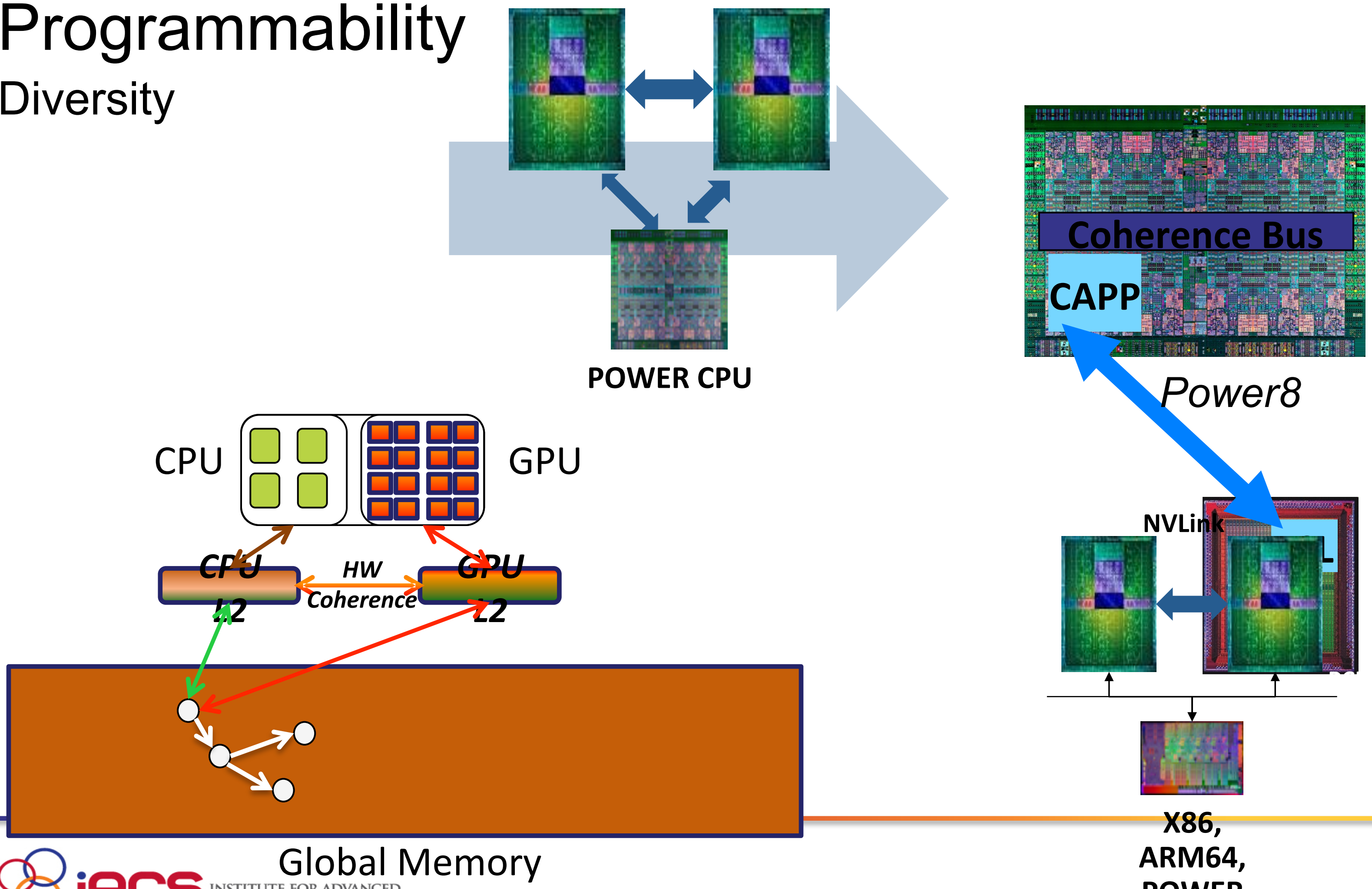
iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

So Will Integration of Accelerators

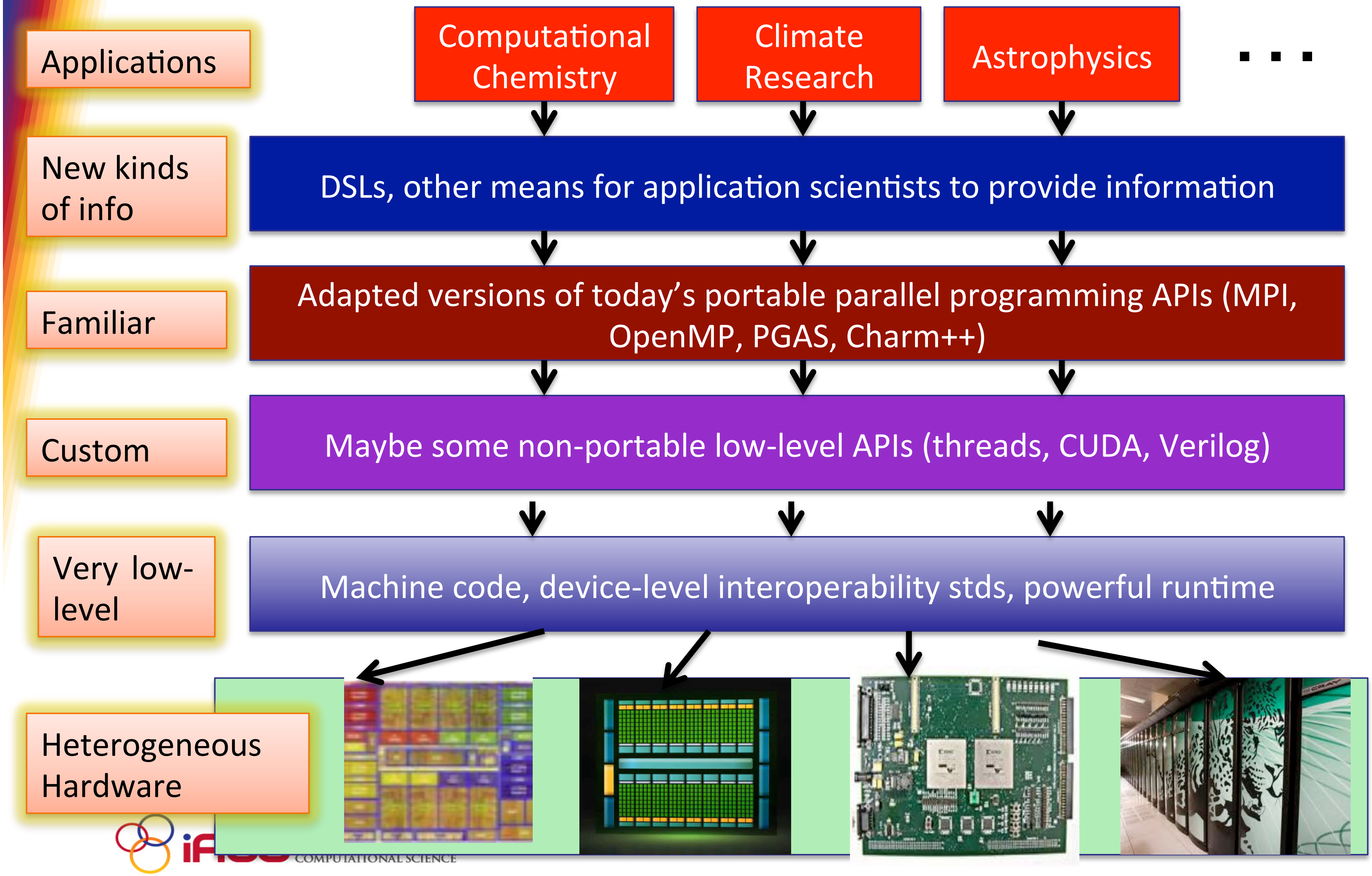
HCA, CAPI, GPU interconnect

Programmability

Diversity



A Layered Programming Approach



Applications

Computational Chemistry

Climate Research

Astrophysics

. . .

New kinds of info

DSLs, other means for application scientists to provide information

Familiar

Adapted versions of today’s portable parallel programming APIs (MPI, OpenMP, PGAS, Charm++)

Custom

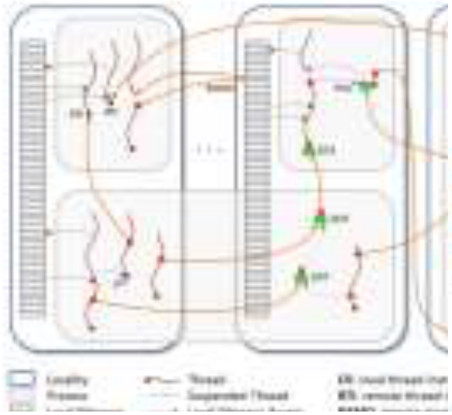
Maybe some non-portable low-level APIs (threads, CUDA, Verilog)

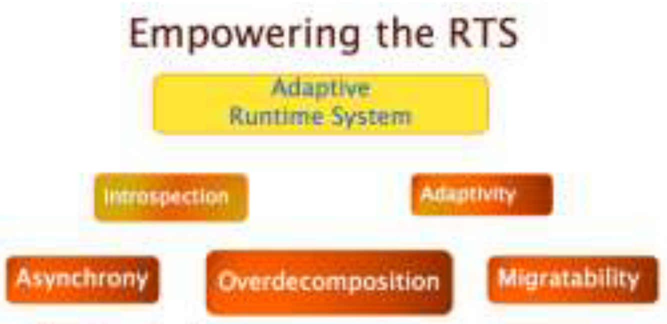
Very low-level

Machine code, device-level interoperability stds, powerful runtime

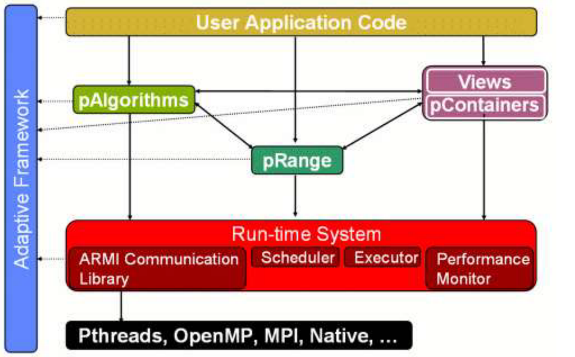
Heterogeneous Hardware

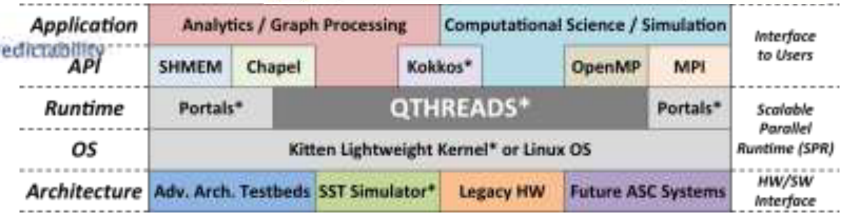
More Dynamic Execution?





- The Adaptive RTS can:
  - Dynamically balance loads
  - Optimize communication:
    - Spread over time, async collectives
  - Automatic latency tolerance
  - Prefetch data with almost perfect predictability





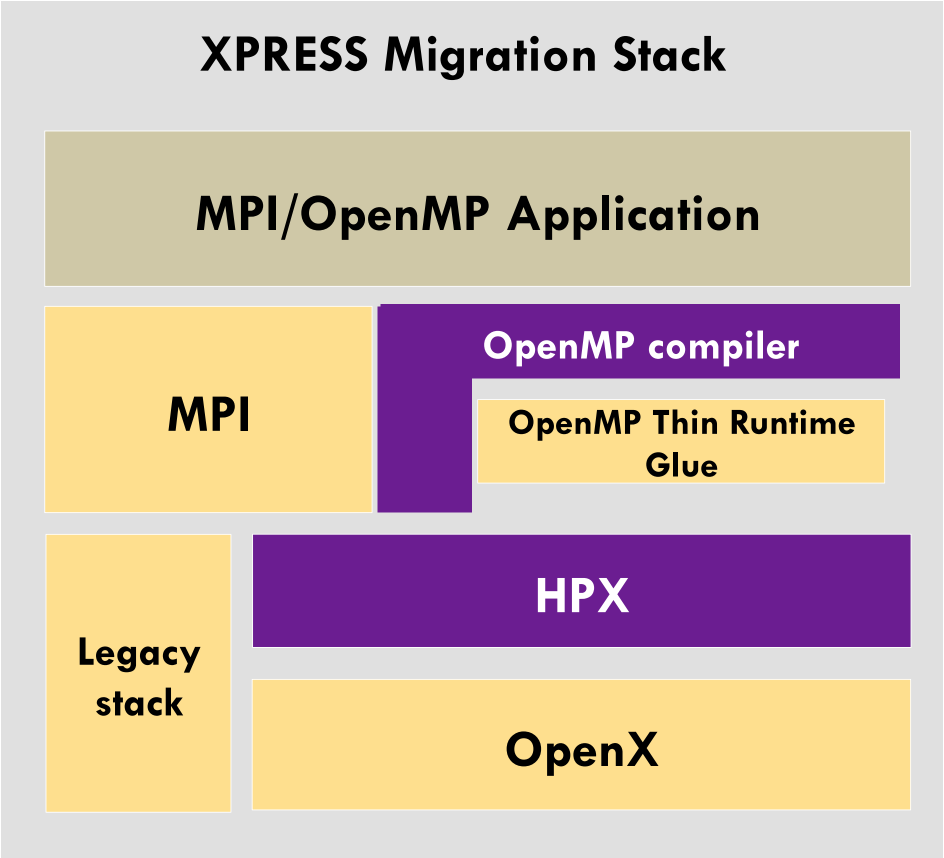
Application	Analytics / Graph Processing			Computational Science / Simulation		Interface to Users
API	SHMEM	Chapel	Kokkos*	OpenMP	MPI	
Runtime	Portals*	QTHREADS*			Portals*	Scalable Parallel Runtime (SPR)
OS	Kitten Lightweight Kernel* or Linux OS					
Architecture	Adv. Arch. Testbeds	SST Simulator*	Legacy HW	Future ASC Systems		HW/SW Interface

- What will the runtime (RT) environment look like? How dynamic will it be?
- Role of runtime system? Relationship between RT and OS, programming models? How is information exchanged?

Performance less predictable in dynamic execution environment

OpenMP in an Exascale World

- *OpenX: prototype* software stack for Exascale systems
  - HPX is runtime system
  - Lightweight threads
  - Thread migration for load balancing, throughput.
- Translating OpenMP -> HPX
  - Maps OpenMP task and data parallelism onto HPX
  - Exploit data flow execution capabilities at scale
  - Big increase in throughput for fine-grained tasks
- Migration path for OpenMP applications



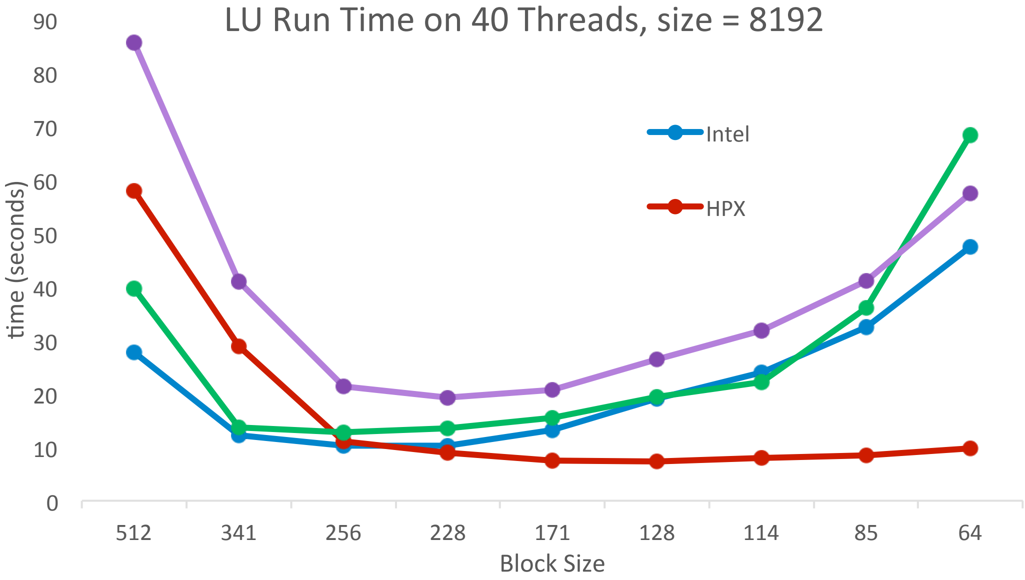
OpenMP over HPX (on-going work)

- Execution model: dynamic adaptive resource management; message-driven computation; efficient synchronization; global name space; task scheduling

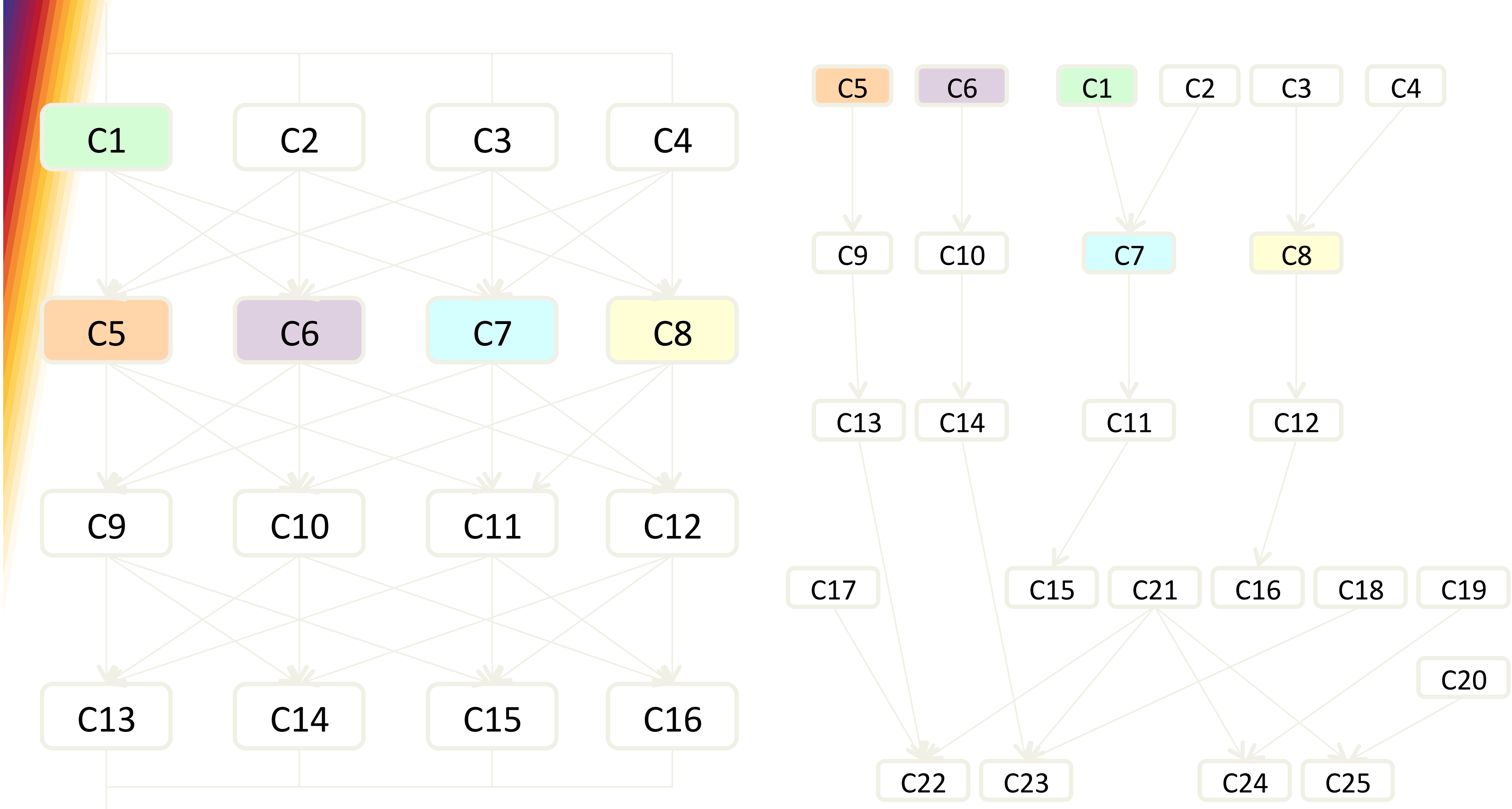
- OpenMP translation:

  No direct interface to OS threads

  - No tied tasks
  - Threadprivate tricky, slow
  - Doesn’t support places
  - OpenMP task dependencies via futures
  - HPX locks faster than OS locks



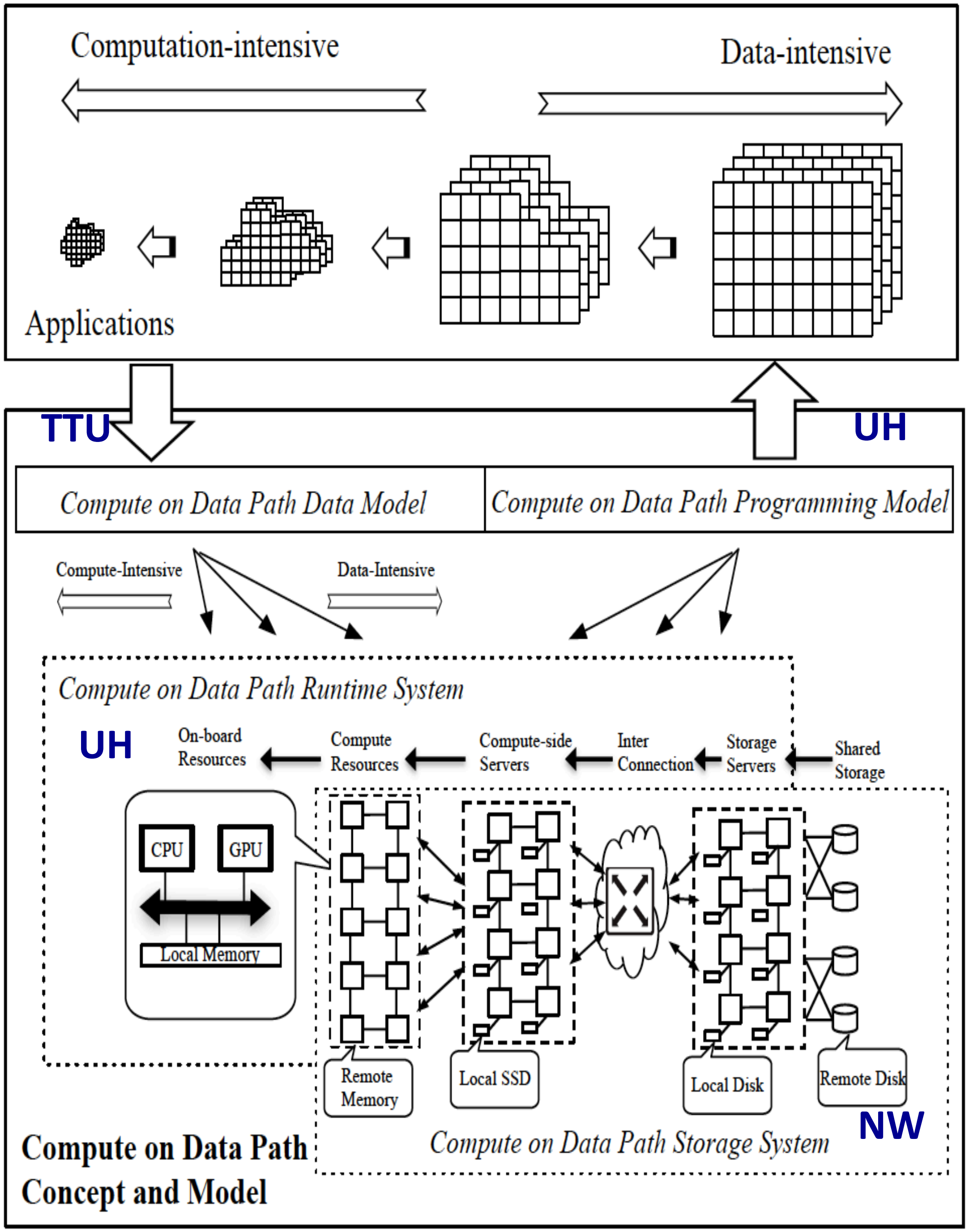
Synchronization in OpenMP Execution



T.-H. Weng, B. Chapman: Implementing OpenMP Using Dataflow Execution Model for Data Locality and Efficient Parallel Execution. Proc. HIPS-7, 2002

A Data-Centric Era

- Continuum of needs from computation-heavy to data heavy
- Potentially within a single application or workflow
- Need to address data movement in its entirety
  - Data Layout
  - New kinds of memory
- What role does user play?



Where are Directives Headed?

- OpenMP has shown significant staying power despite some big changes in hardware characteristics
  - Broad user base; yet strong HPC representation
  - Paying more attention to data locality, affinity, tasking
- Need to continue to evolve directives and implementation
  - Data and memory challenges remain
  - Less synchronization, more tasks, is good
  - Performance; validation, power/energy savings,..
  - Runtime: resources, more dynamic execution
- What about level of abstraction?
  - Performance portability is a major challenge
  - OpenMP codes often hardwire in system-specific details

Wrap-Up

- Programmers need portable, productive programming interfaces
  - Directives help deliver new concepts
  - Hardware changes require us to continue to adapt
  - Importance of accelerator devices likely to grow
  - Many new challenges posed by diversity, large data sets, memory and new application trends
- Directives pretty successful

- Not all the answers are in the programming interface
  - New or adapted algorithms
  - Novel compiler translations; modeling for smart decisions
  - Innovative implementations and runtime adaptations
  - Tools to facilitate development and tuning