An Evolutionary Exascale Programming Model Deserves Revolutionary Support

Barbara Chapman

University of Houston

HIPS '12, Shanghai, 5/21/2012

Acknowledgements: NSF CNS-0833201, CCF-0917285;
DOE DE-FC02-06ER25759

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

Agenda

- Emerging HPC Architectures and their Programming Models
- OpenMP: An Evolutionary Approach to Node Programming
- Some Language Ideas for Locality
- Compiler Efforts: Increasing the Benefits
- Runtime and Tool Support

Petascale is a Global Reality

- K computer
  - 68,544 SPARC64 VIIIfx processors, Tofu interconnect, Linux-based enhanced OS, produced by Fujitsu
- Tianhe-1A
  - 7,168 Fermi GPUs and 14,336 CPUs; it would require more than 50,000 CPUs and twice as much floor space to deliver the same performance using CPUs alone.
- Jaguar
  - 224,256 x86-based AMD Opteron processor cores, Each compute node features two Opterons with 12 cores and 16GB of shared memory
- Nebulae
  - Nvidia Tesla 4640 GPUs, Intel X5650-based 9280 CPUs
- Tsubame
  - 4200 GPUs

Exascale Systems: The Planning

- **Town Hall Meetings April-June 2007**
- **Scientific Grand Challenges Workshops November 2008 – October 2009**
  - Climate Science, High Energy Physics, Nuclear Physics, Fusion Energy, Nuclear Energy, Biology, Material Science and Chemistry, National Security (with NNSA)
- **Cross-cutting workshops**
  - Architecture and Technology (12/09)
  - Architecture, Applied Mathematics and Computer Science (2/10)
- **Meetings with industry (8/09, 11/09)**
- **External Panels**
  - ASCAC Exascale Charge
  - Trivelpiece Panel

Peak performance is 10**18 floating point operations per second

- **International Exascale Software Project (IESP) (2010 – 2012)**
  - International effort to specify research agenda that will lead to exascale capabilities
  - Academia, labs, agencies, industry
  - Focused meetings to determine R&D needs, foster international collaboration
  - Significant contribution of open-source software
  - Produced a detailed roadmap

IESP: Exascale Systems

Given budget constraints, predictions focused on two alternative designs:

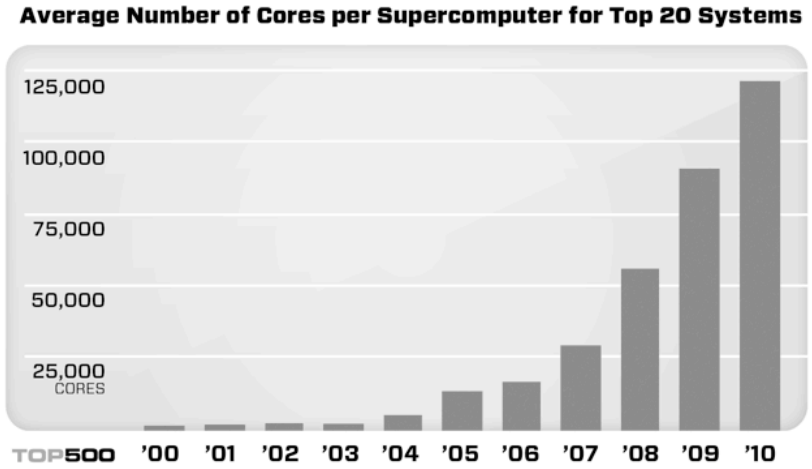
- Huge number of lightweight processors, e.g. 1 million chips, 1000 cores/chip = 1 billion threads of execution
- Hybrid processors, e.g. 1.0GHz processor and 10000 FPUs/ socket & 100000 sockets/system = 1 billion threads of execution

Other predictions made in 2010:

- Modest increase in number of nodes in system
- Operational cost prohibitive unless power greatly reduced
- Exascale platforms expected to arrive around 2018

See http:/www.exascale.org/

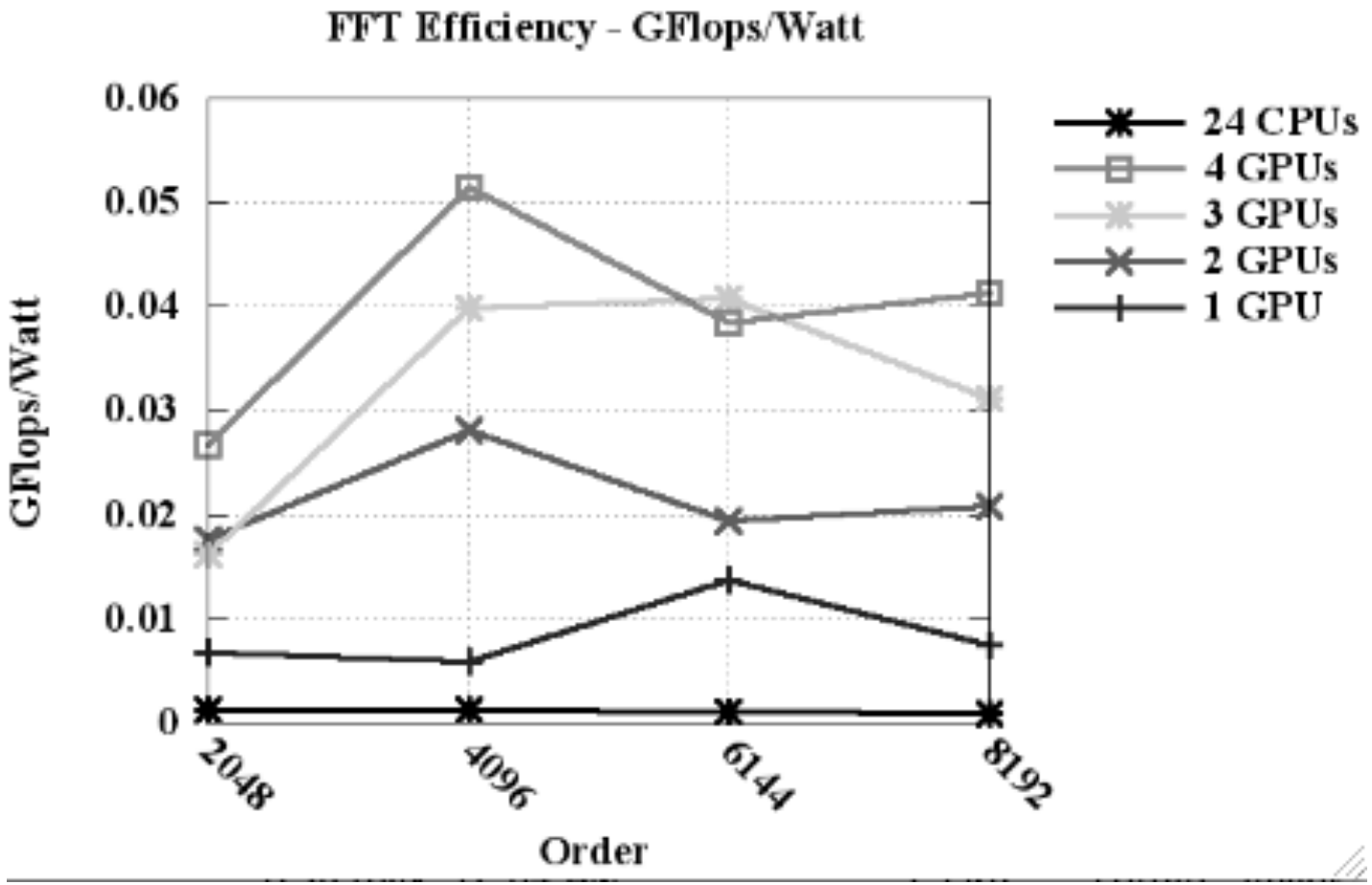
Exascale: Anticipated Architectural Changes



- Massive (ca. 4X) increase in concurrency
  - **Mostly within compute node**
- Balance between compute power and memory changes significantly
  - 500x compute power and 30x memory of 2PF HW
  - Memory access time lags further behind

Biggest change for HPC since distributed memory systems introduced

FFT – Energy Efficiency



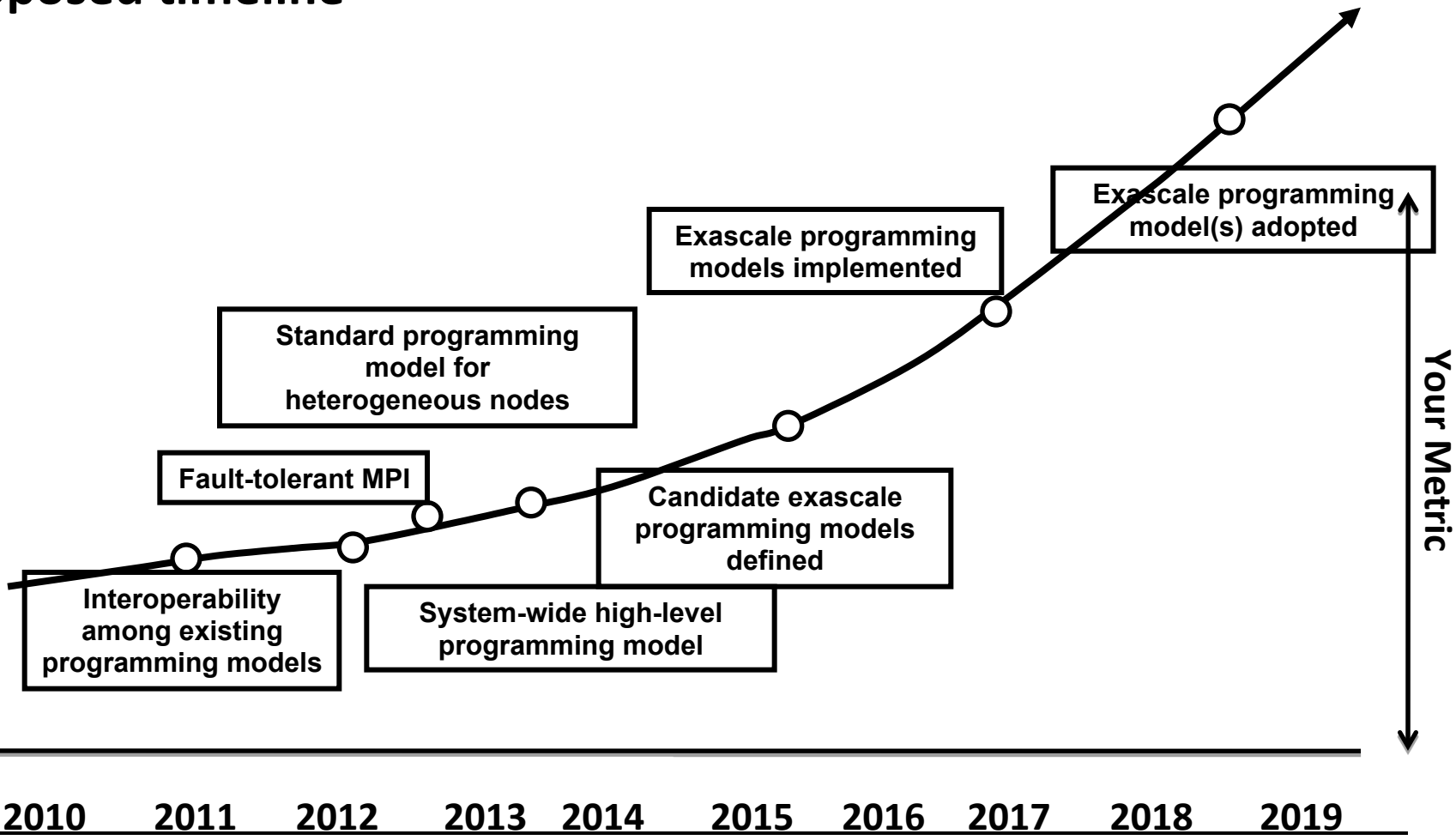
Programming Challenges

- Architecture/software co-design must address
  - Scalability, memory savings, power efficiency
  - Design and use of exascale I/O systems
  - System resilience and fault tolerant apps
  - Potential heterogeneity in node
  - Levels of parallelism
- What is the programming model?
  - Performance, portability, productivity
  - Evolution or revolution?

IESP Programming Models

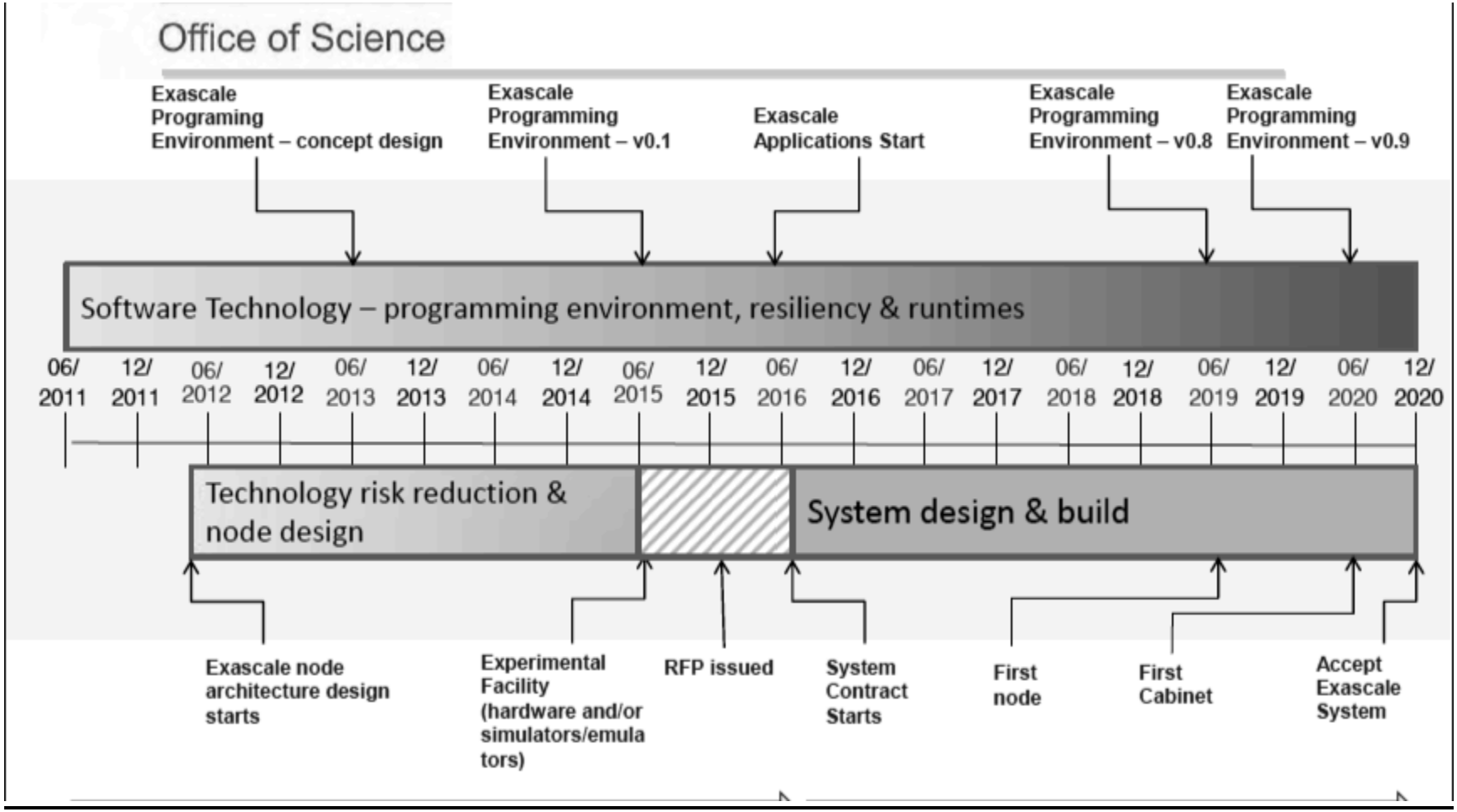
International Exascale Software Project

Proposed timeline



www.exascale.org

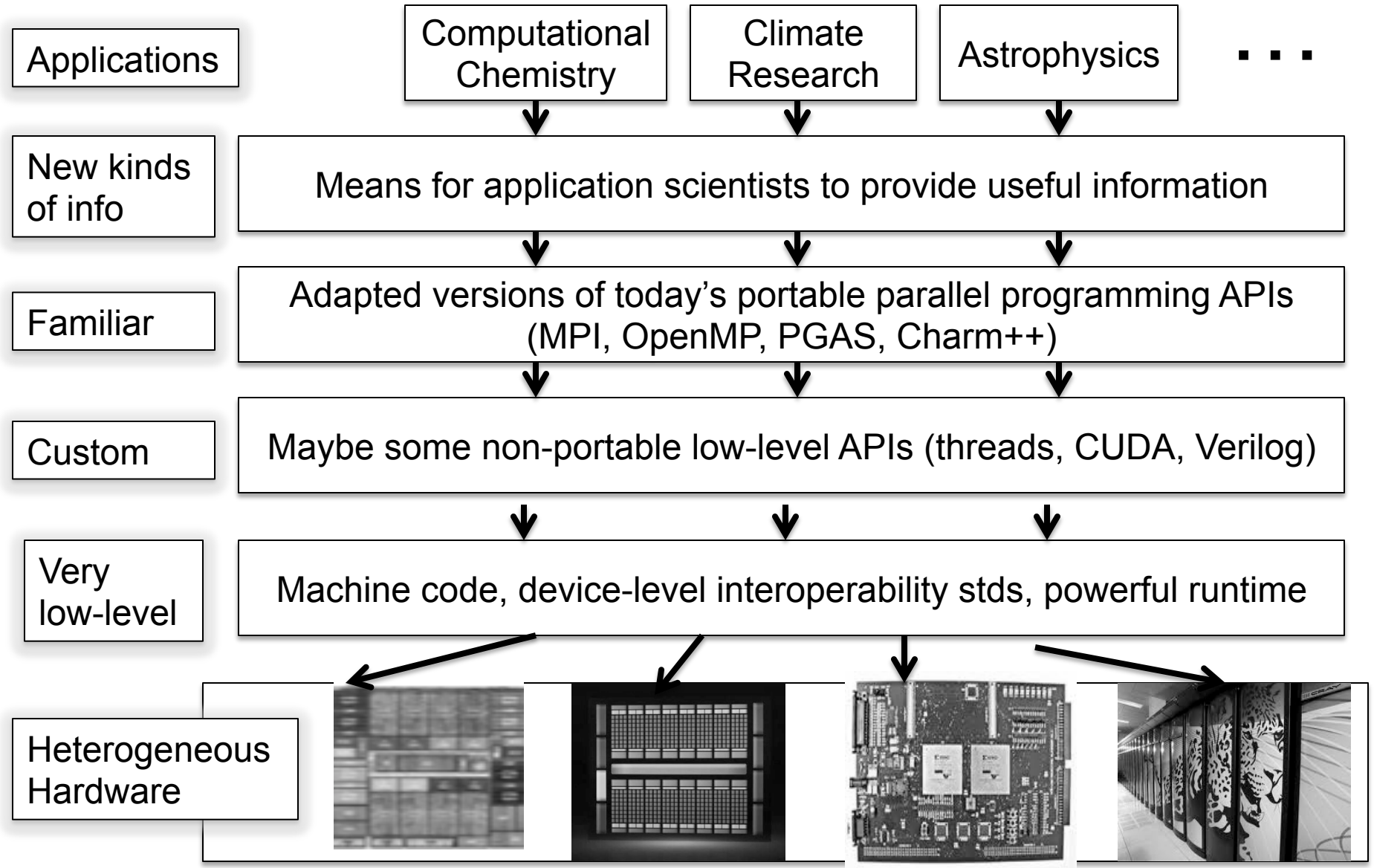
DOE Workshop's Reverse Timeline



Evolution or Revolution?

- Timing of programming model delivery is critical
  - Must be in place when machines arrive
  - Needed earlier for development of system software, new codes
- Evolutionary approach as baseline
  - Most likely to work, easiest adaptation for existing code
  - MPI and OpenMP most likely candidate
- Higher levels of abstraction could be, initially, mapped to evolutionary solution
  - Layers of programming models with different kinds of abstractions
  - Higher level programming model subject of intense research

A Layered Programming Approach



Agenda

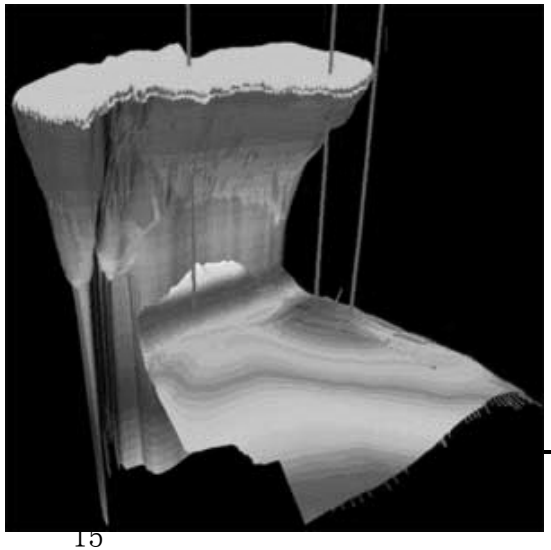
- Emerging HPC Architectures and their Programming Models
- OpenMP: An Evolutionary Approach to Node Programming
- Some Language Ideas for Locality
- Compiler Efforts: Increasing the Benefits
- Runtime and Tool Support

The OpenMP ARB 2011

- 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, CAPS Entreprise, Cray, Fujitsu, HP, IBM, Intel, Microsoft, NEC, Nvidia, Oracle, PGI, Texas Instruments
    - Auxiliary: ANL, cOMPunity, EPCC, NASA, LANL, LLNL, ORNL, RWTH Aachen, TACC

The OpenMP Shared Memory API

- High-level directive-based multithreaded programming
  - User makes strategic decisions; compiler figures out details
  - Use on node can reduce memory footprint, communication behavior of MPI code
  - Already being used with MPI in DOE application codes
  - **Does not directly address locality, heterogeneous nodes**



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

GPU (Energy Cost Per Ops)

Operation	Energy (pJ)	DP FLOPs	Insts*
I$ Fetch	33	0.67	2.0
Register Access (3W)	10.5	0.2	0.6
Access 3 D$	100	2	6
Access 3 L2 D$	460	9	27
Access 3 off chip	762	15	45
Access 3 from DRAM	6000	120	360

http://www.lbl.gov/cs/html/Manycore_Workshop09/GPU%20Multicore%20SLAC%202009/dallyppt.pdf

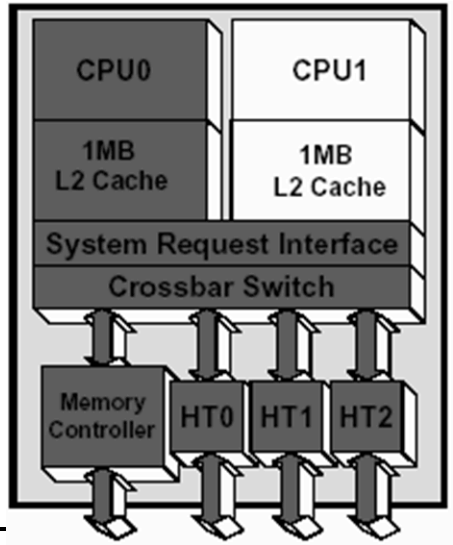
GPU (Energy Cost Per Ops)

Operation	Energy (pJ)	DP FLOPs	Insts*
I$ Fetch	33	0.67	2.0
Register Access (3W)	10.5	0.2	0.6
Access 3 D$	100	2	6
Access 3 L2 D$	460	9	27
Access 3 off chip	762	15	45
Access 3 from DRAM	6000	120	360

http://www.lbl.gov/cs/html/Manycore_Workshop09/GPU%20Multicore%20SLAC%202009/dallyppt.pdf

OpenMP and Data Locality

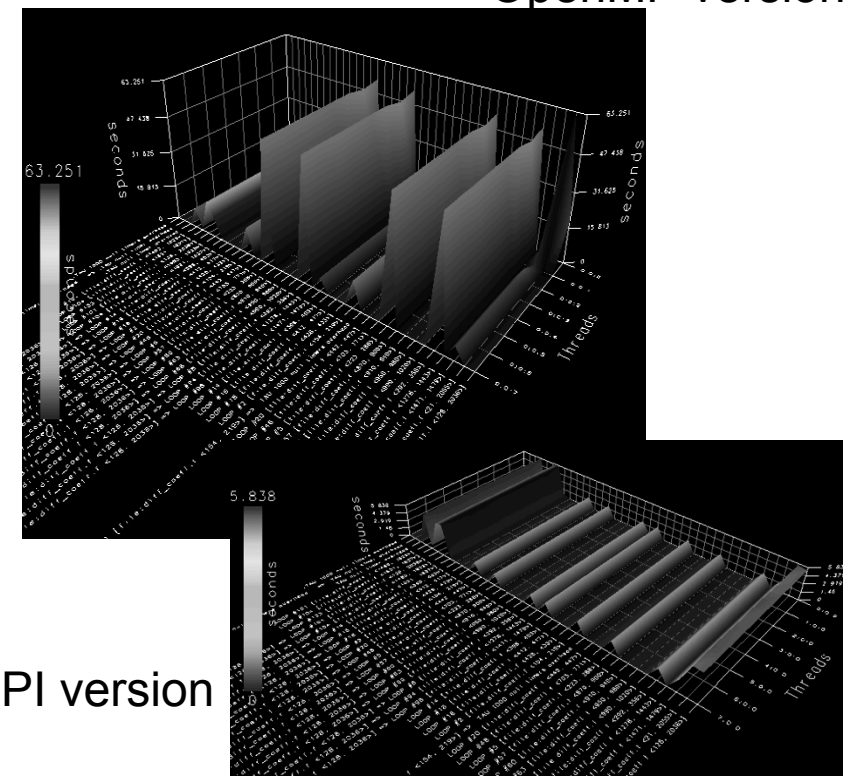
- OpenMP does not permit explicit control over data locality
- Thread fetches data it needs into local cache
- Implicit means of data layout popular on NUMA systems
  - As introduced by SGI for Origin
  - “First touch”
- Emphasis on privatizing data where possible, and optimizing code for cache
  - This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



Small "Mistakes", Big Consequences

- GenIDLEST
  - Scientific simulation code
  - Solves incompressible Navier Stokes and energy equations
  - MPI and OpenMP versions
- Platform
  - SGI Altix 3700 (NUMA)
  - 512 Itanium 2 Processors
- OpenMP code slower than MPI

OpenMP version

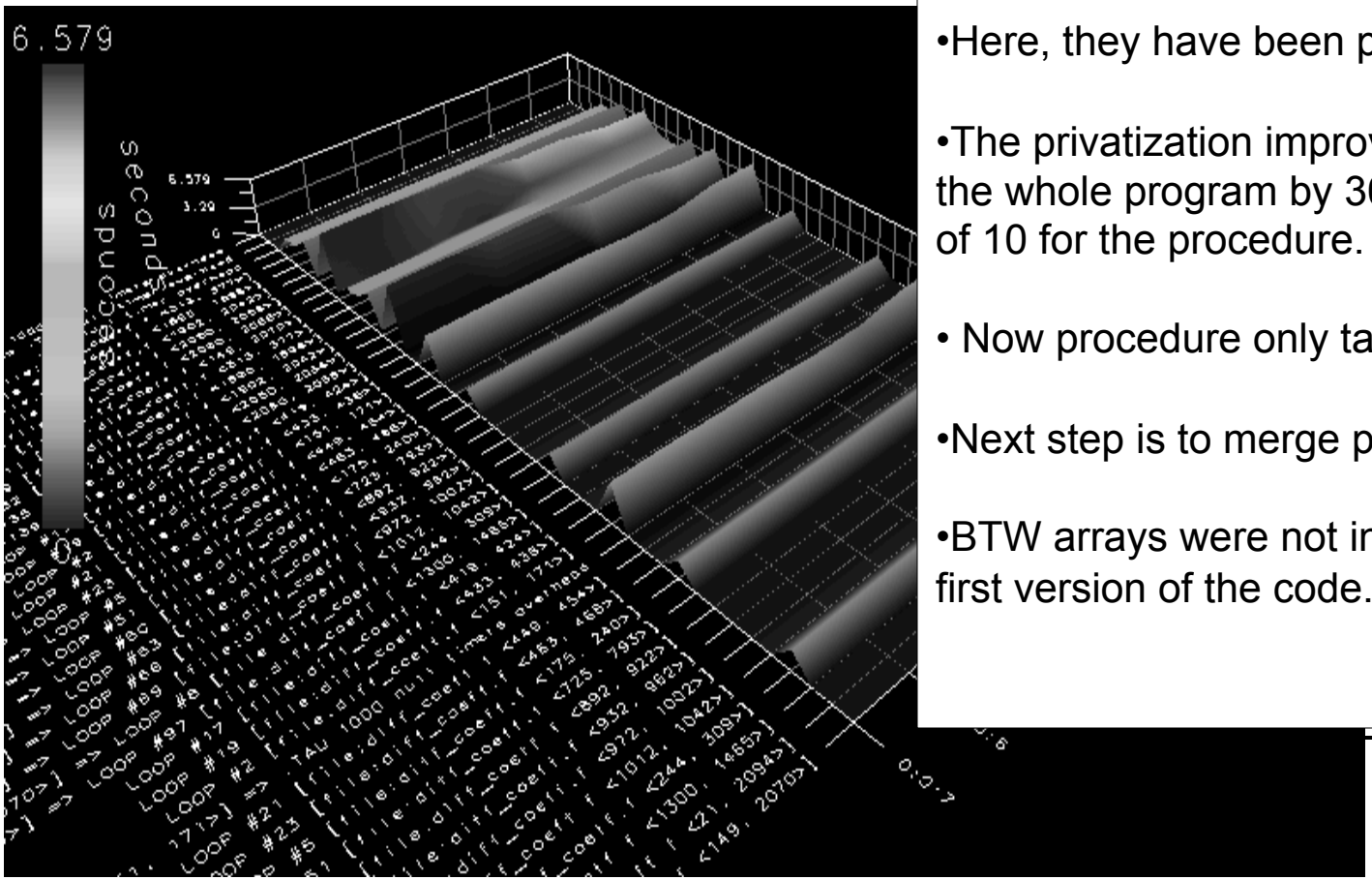


MPI version

In the OpenMP version , a single procedure is responsible for 20% of the total time and is 9 times slower than the MPI version . Its loops are up to 27 times slower in OpenMP than MPI.

A Solution: Privatization

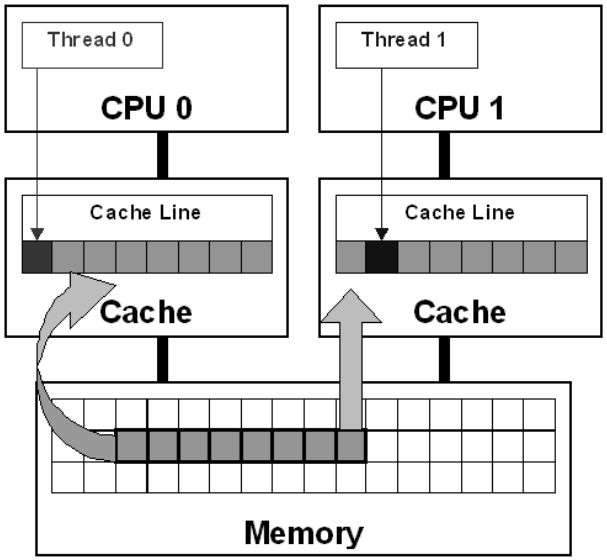
OpenMP Optimized Version



- Lower and upper bounds of arrays used privately by threads are shared, stored in same memory page and cache line
- Here, they have been privatized.
- The privatization improved the performance of the whole program by 30% and led to a speedup of 10 for the procedure.
- Now procedure only takes 5% of total time
- Next step is to merge parallel regions..
- BTW arrays were not initialized via first touch in first version of the code.

Effects of False Sharing

False sharing is a performance degrading data access pattern that can arise in systems with distributed, coherent caches.

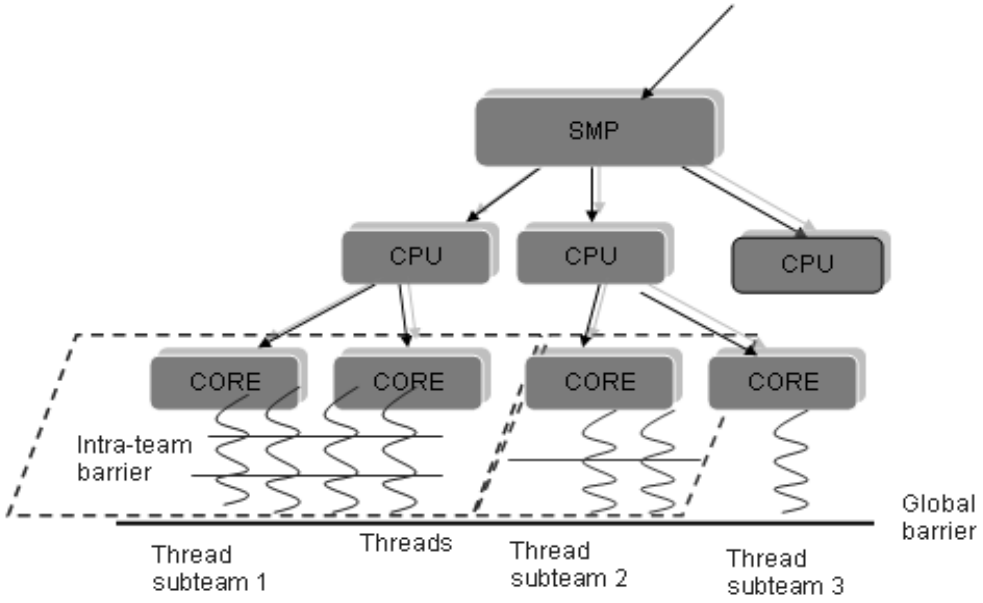


Code Version	Execution Time (sec)			
	Sequential	2 threads	4 threads	8 threads
Unoptimized	0.503	4.563	3.961	4.432
Optimized	0.503	0.263	0.137	0.078

Agenda

- Emerging HPC Architectures and their Programming Models
- OpenMP: An Evolutionary Approach to Node Programming
- Some Language Ideas for Locality
- Compiler Efforts: Increasing the Benefits
- Runtime and Tool Support

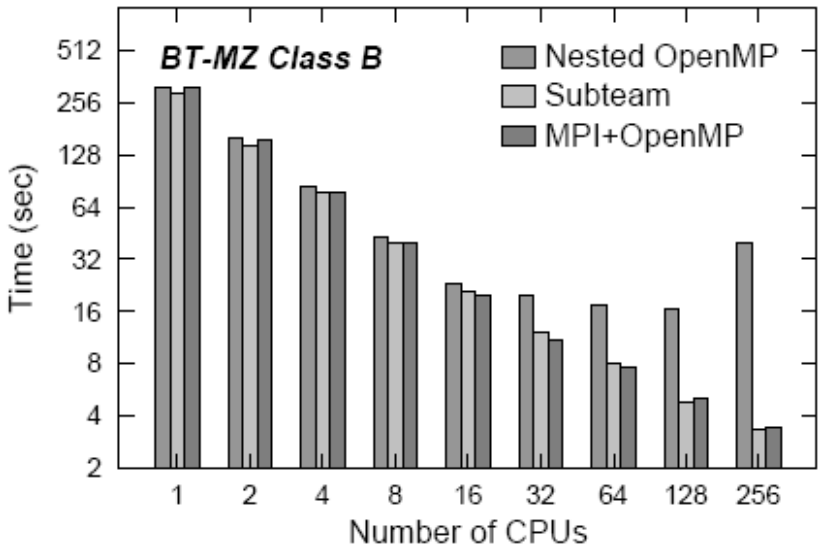
Subteams of Threads?



Thread Subteam: subset of threads in a team

- Overlap computation and communication (MPI)
- Concurrent worksharing regions
- Additional control of locality of computations and data
- Handle loops with little work

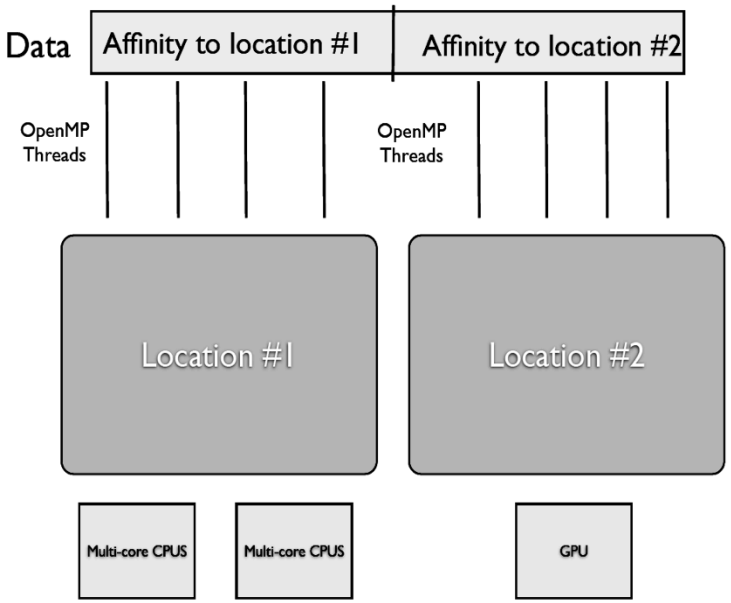
```
for (j=0; j< ProcessingNum; j++)
        #pragma omp for schedule(dynamic)
        subteam(2:omp_get_num_threads()-1)
          for (k=0; k<M; k++) {
            ProcessData(); // data processing
          } // subteam-internal barrier
```



Increases expressivity of single-level parallelism

OpenMP Locality Research

Locations := Affinity Regions, Based on Locales, Places

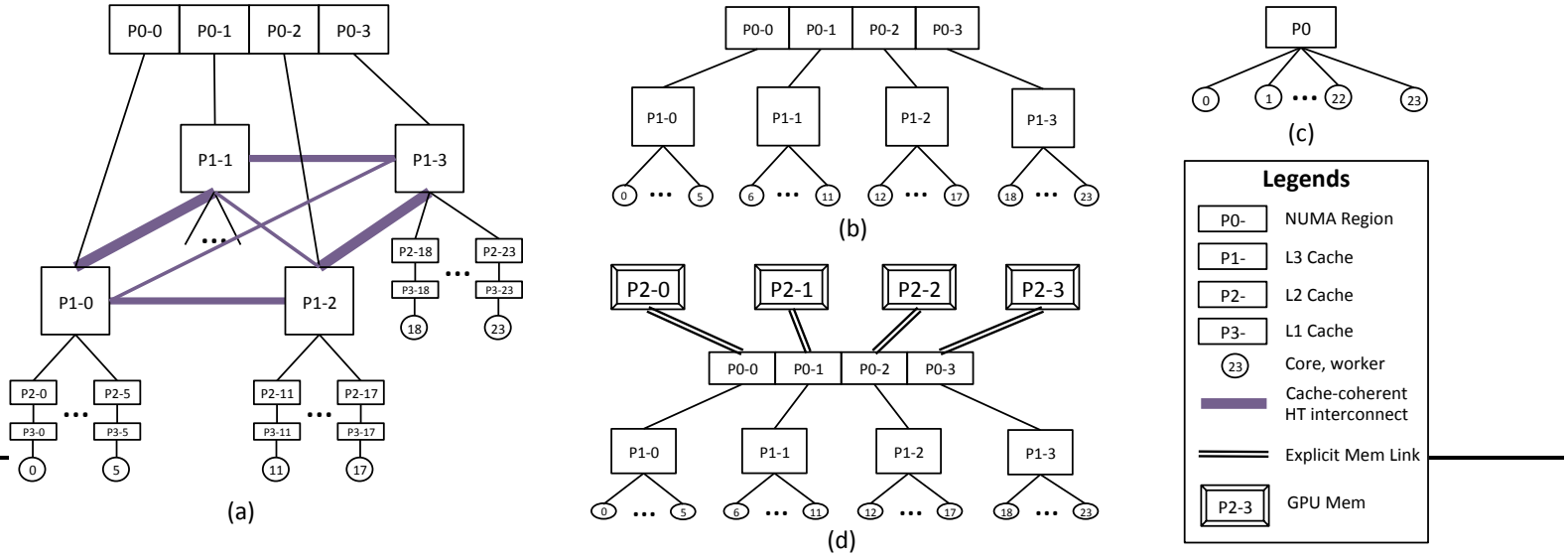


- Means to manage data layout and enhance locality.
- Adapts Chapel/X10 ideas
  - Represent execution environment by collection of “locations”
  - Map data, threads to a location; distribute data across locations
  - Align computations with data’s location, or map them explicitly
- Significant performance boost on mid-size SMP systems.

Lei Huang, Haoqiang Jin, Barbara Chapman, Liqi Yi. Enabling Locality-Aware Computations in OpenMP. Scientific Computing, Vol 18, Numbers 3-4, 169-181, IOS Press Amsterdam, 2010

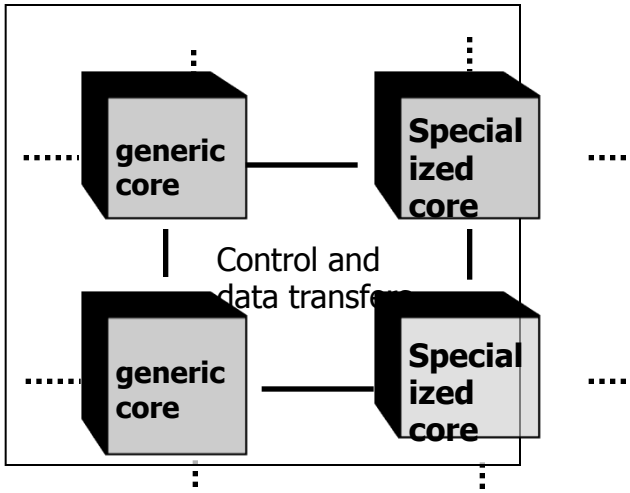
Hierarchical "Place" Trees Abstraction

- Solutions to **locality**, hw modeling and implicit/explicit **data movement**
  - Memory modules (Mem, NUMA region, caches, etc)→places, cores → workers
- Program Machine Tree
  - Programmer view, a tree. Default: just one place (mem+cores)
  - APIs for accessing an HPT, for placing data and binding tasks with data
- Platform Machine Tree
  - Compiler and runtime view
  - Machine aware compilation, and runtime adaptation



A Heterogeneous World

- OpenMP could be the basis for a unified, productive programming model for heterogeneous nodes
  - How to identify code that should run on a certain kind of core?
  - Where and when is data allocated?
  - How to optimize data motion?



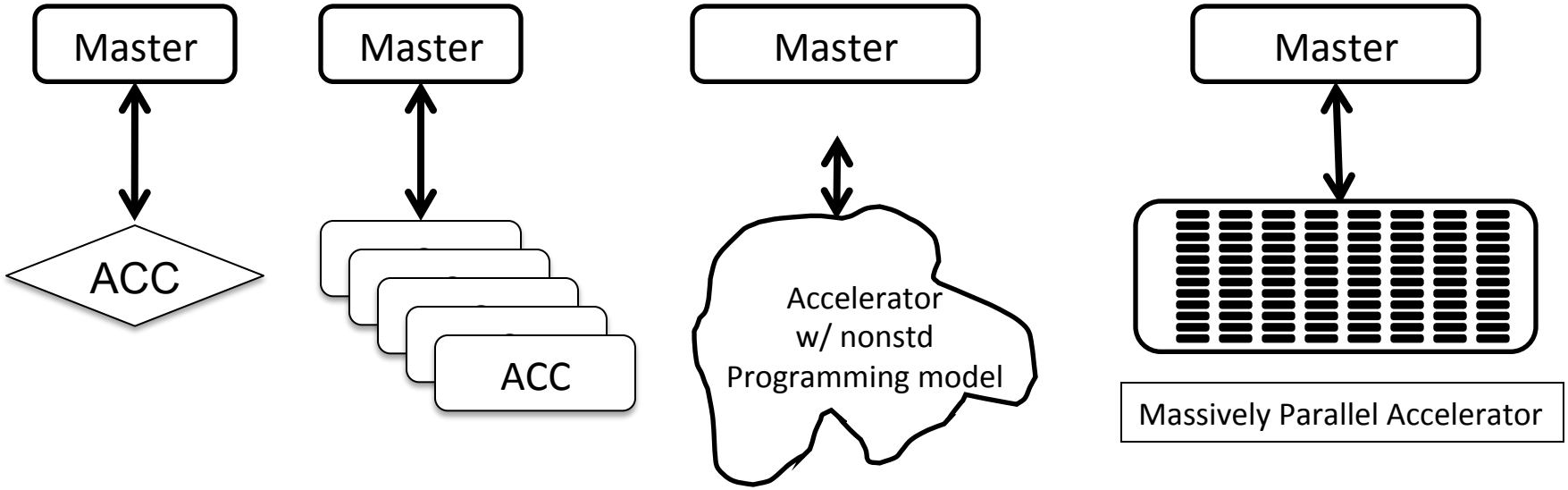
HMPP

```
//acquire a device
#pragma hmpp fdtd acquire
//allocate data on the device
#pragma hmpp fdtd allocate
#pragma hmpp fdtd allocate, data["in";"out"], data["in";"out"].size={dx*dy*dz}, &
#pragma hmpp &              data["in";"out"].elementsize="sizeof(double)"
//upload of data based on the address - mirroring
#pragma hmpp fdtd advancedload, data["in";"out"]
#pragma hmpp fdtd callsite
FDTD_base (in, out, dx, dy, dz, c[0], c[1], c[2],
//download of data based on the address
#pragma hmpp fdtd delegatedstore, data["out"]
// deallocation of data mirror
#pragma hmpp fdtd free, data["in";"out"]
#pragma hmpp fdtd release
```

```
#pragma acc region for parallel copyin (V[0:(dx*dy*dz)]) copy (U[0:(dx*dy*dz)])
{
#pragma acc for independent
for (k = 4; k < dz-4; k++)
{
        #pragma acc for independent
        for (j = 4; j < dy-4; j++)
        {
                #pragma acc for independent
                for (i = 4; i < dx-4; i++)
                {
```

PGI

Heterogeneity in OpenMP 4.0 Attempts To Target Range of Acceleration Configurations



- Dedicated hardware for specific function(s)
  - Attached to a master processor
  - Multiple types or levels of parallelism
    - Process level, thread level, ILP/SIMD
- May not support a full C/C++ or Fortran compiler
  - May lack stack or interrupts, may limit control flow, types

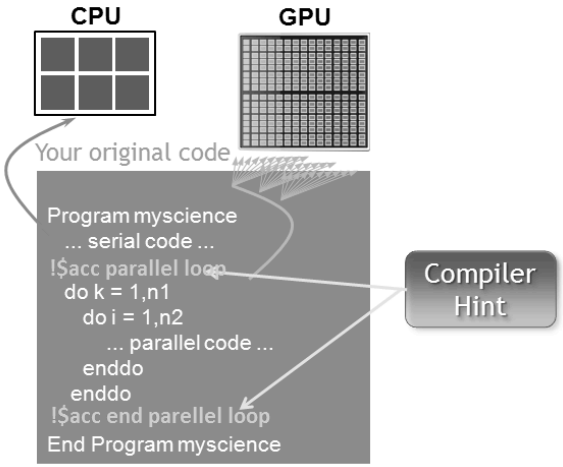
OpenACC came from this on-going effort

Control Locality of Work and Data

```
void foo(double A[], double B[], double C[], int nrows, int ncols) {
#pragma omp data_region acc_copyout(C), host_shared(A,B)
  {
    #pragma omp acc_region
    for (int i=0; i < nrows; ++i)
      for (int j=0; j < ncols; j += NLANES)
        for (int k=0; k < NLANES; ++k) {
          int index = (i * ncols) + j + k;
          C[index] = A[index] + B[index];
        } // end accelerator region
    print2d(A,nrows,ncols);
    print2d(B,nrows,ncols);
    Transpose(C); // calls function w/another accelerator construct
  } // end data_region
  print2d(C, nrows, ncols);
}
void Transpose(double X[], int nrows, int ncols) {
   #pragma omp acc_region acc_copy(X), acc_present(X)
   { … }
}
```

OpenACC

- Compiler directives that specify loops and regions of code to be offloaded from a host CPU to an attached accelerator
- Fine-grained control over allocation of variables and copying of data
  - Compiler creates kernels
  - C, C++ and Fortran bindings
- Provides portability across operating systems, host CPUs and accelerators.
- Members – PGI, Cray, NVIDIA, CAPS
- OpenACC V 1.0 specification – http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf
- http://www.openacc-standard.org/

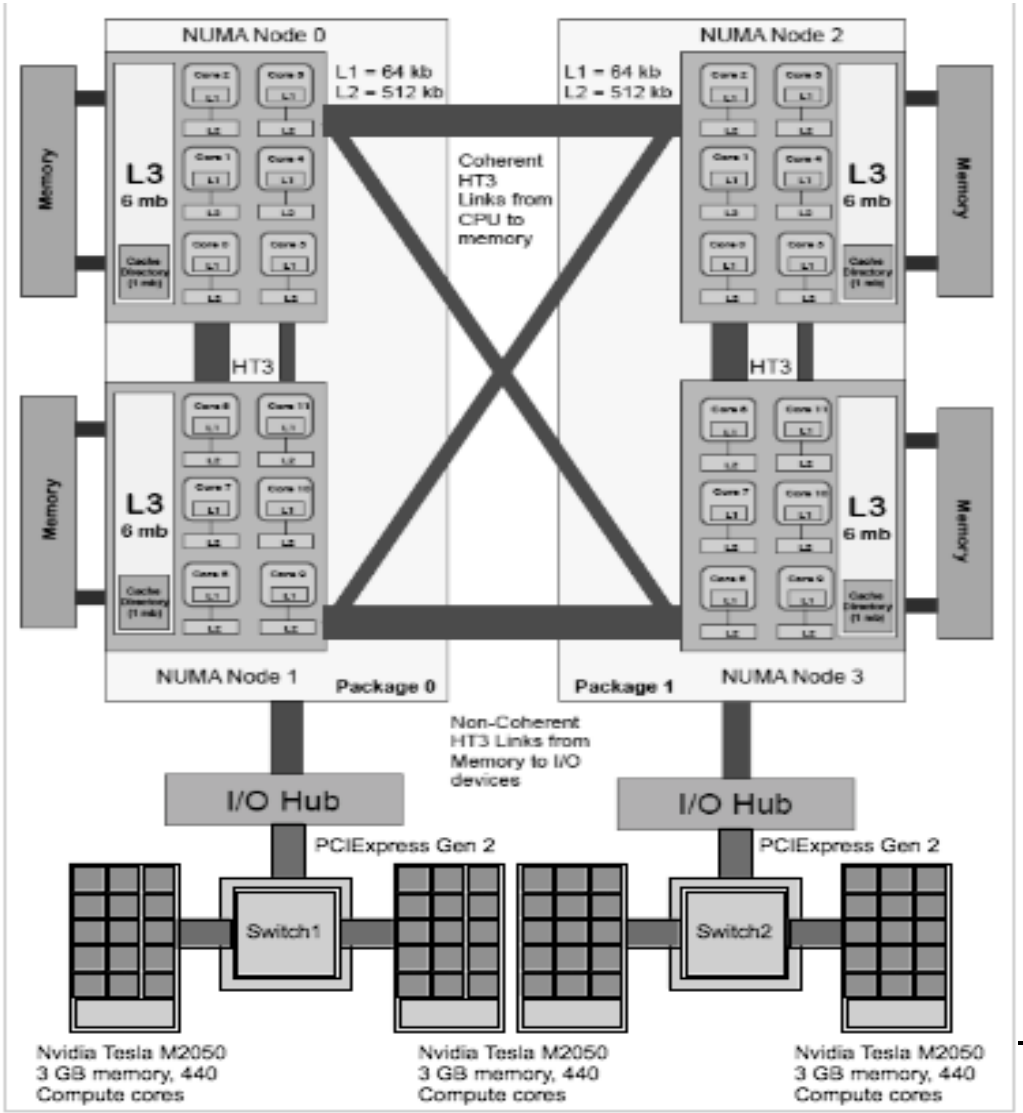


CAPS CRAY THE SUPERCOMPUTER COMPANY NVIDIA The Portland Group

Agenda

- Emerging HPC Architectures and their Programming Models
- OpenMP: An Evolutionary Approach to Node Programming
- Some Language Ideas for Locality
- Compiler Efforts: Increasing the Benefits
- Runtime and Tool Support

Representing Node Architectures

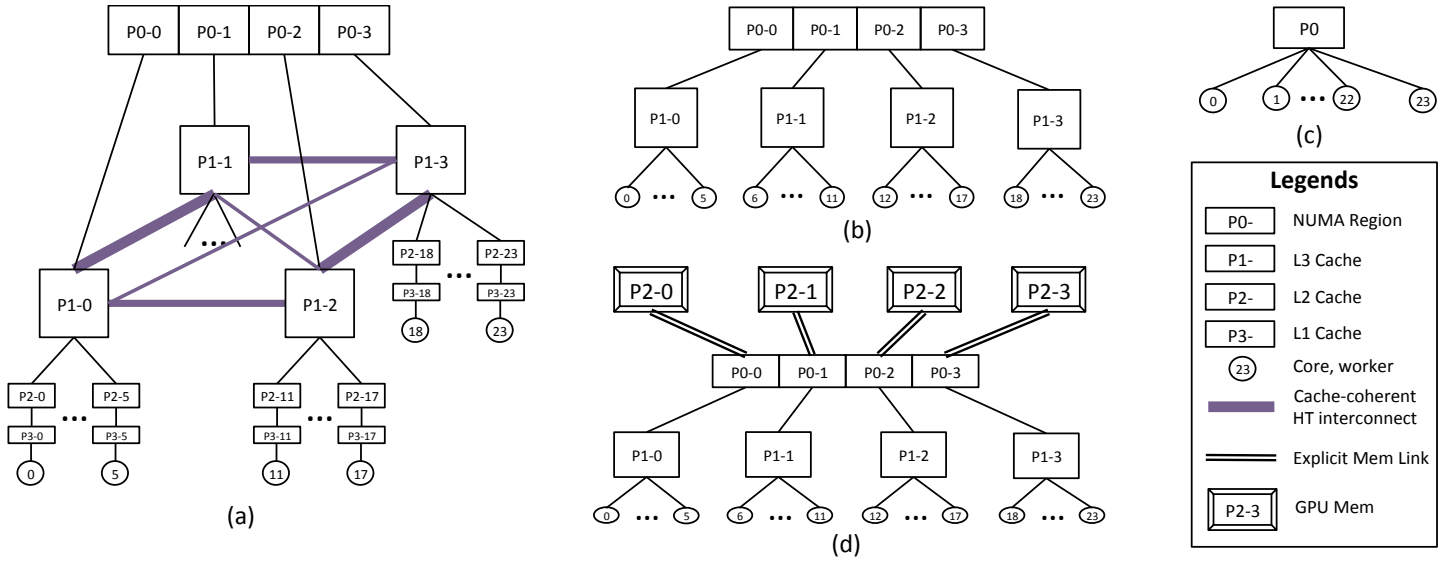


4 2-way AMD Opteron 6174 Magny-Cours processor (24 physical cores)

4 Nvidia Tesla M2050 GPUs (440 compute cores), 3GB GDDR5

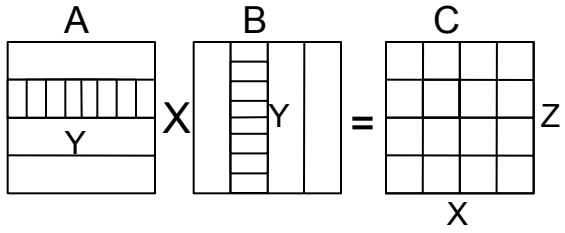
Compiler View of Target Platform

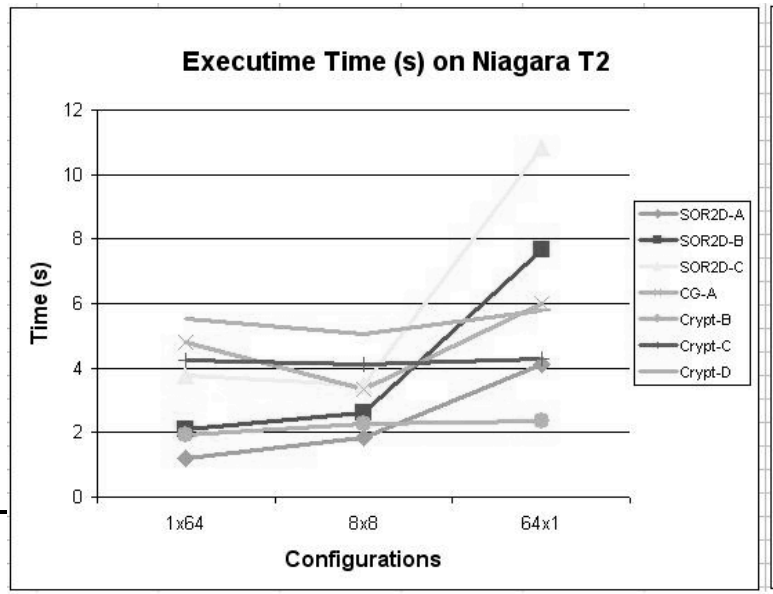
- Platform Machine Tree
  - Compiler and runtime view
  - Machine aware compilation, and runtime adaptation

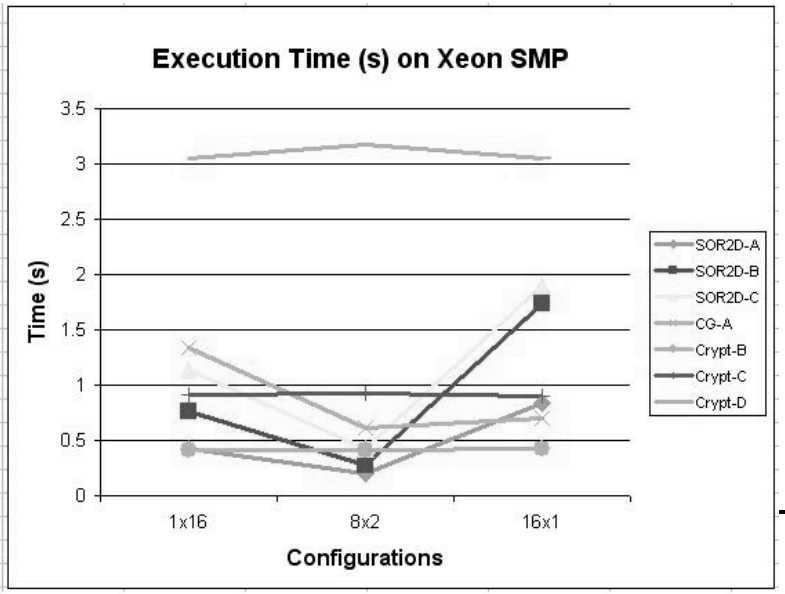


HPT Preliminary Results

```
1   void MatrixMult(double[.] A, double[.] B, double[.] C) {
2       if ( here.isLeafPlace( ) ) { /* compute the sub-block sequentially */
3         for (point [i,j,k] : [A.region.rank(0), B.region.rank(1), A.region.rank(1)])
4           C[i,j] += A[i,k] * B[k,j];
5       }
6       else {
7         /* retrieve children places and structure them into a 2-D Cartesian topology, pTop */
8         CartTopology pTop = here.getCartTopology( 2 );
9
10        /* block distribute C over the 2-D topology, pTop*/
11        final dist C_d = dist.block( C, pTop );
12
13        /* distribute rows of A in accordance with pTop's first dimension */
14        final dist A_d = dist.block( A, pTop, 0 );
15        /* distribute columns of B in accordance with pTop's second dimension */
16        final dist B_d = dist.block( B, pTop, 1 );
17
18        /* recursive call with sub-matrices of A, B, C projected on to place p */
19        finish ateach( point p : pTop ) MatrixMult( A_d|p, B_d|p, C_d|p );
20    }
21  }
```



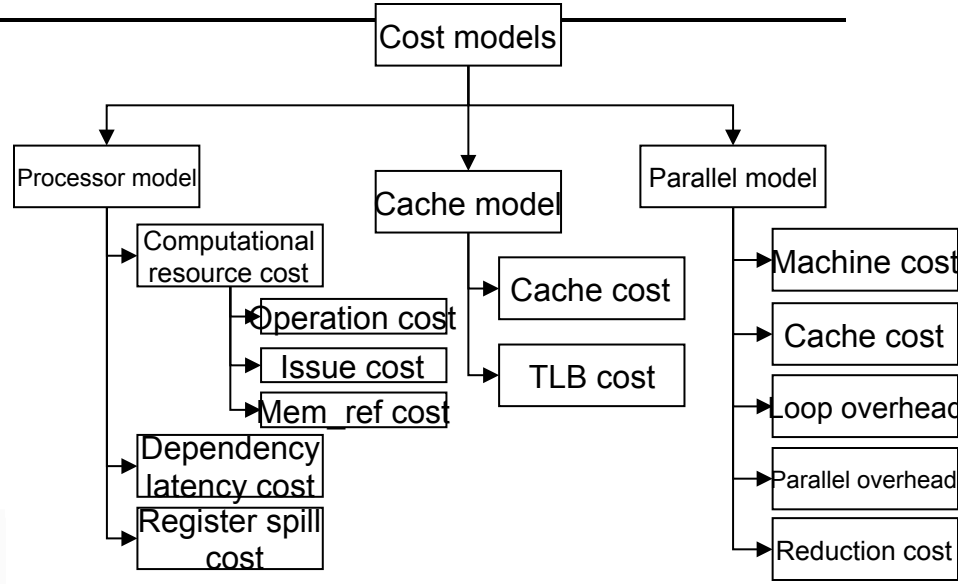


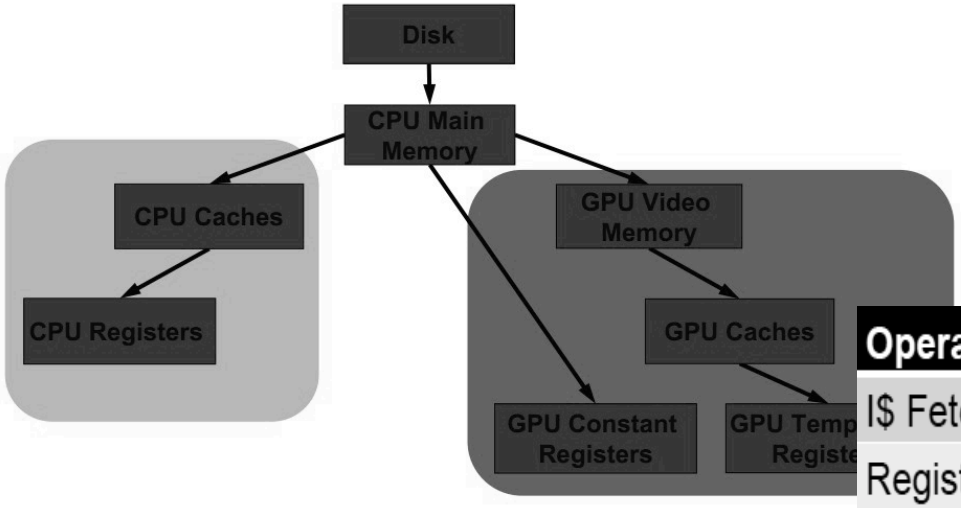


Compiler Modeling and Prediction to Guide Translation and Optimization

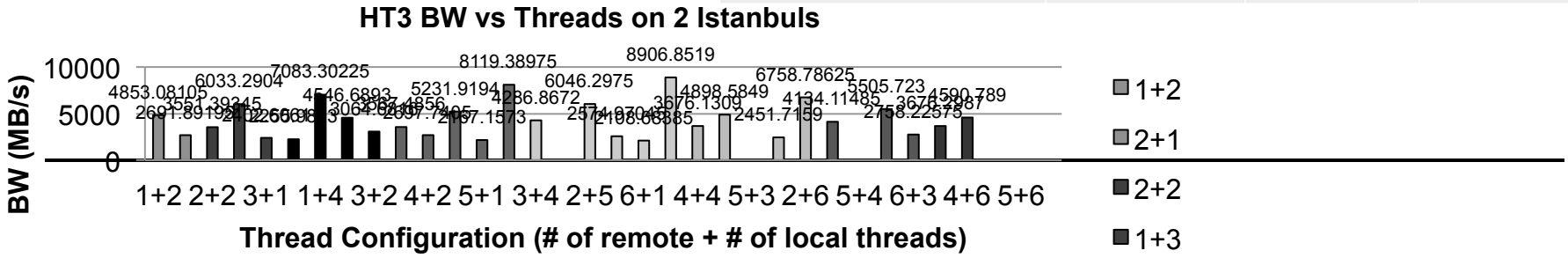
- Conventional approach
  - Mostly evaluates cache effects of uniprocessors
- Taking account of sharing and contention effects
  - Needed on multi- and many-core architectures
  - Consideration of the memory hierarchy structure
  - False sharing, shared cache contention, and memory bandwidth contention and latency
- Consideration of node complexity
  - Multiple kinds of cores, interconnect, structure of memory hierarchies
- Support compile-time and runtime optimization
  - Data placement and affinity between tasks and data
  - Mapping task graphs to the hardware architectures
  - Guided energy-aware scheduling

What to Model?





Operation	Energy (pJ)	DP FLOPs	Insts*
I$ Fetch	33	0.67	2.0
Register Access (3W)	10.5	0.2	0.6
Access 3 D$	100	2	6
Access 3 L2 D$	460	9	27
Access 3 off chip	762	15	45
Access 3 from DRAM	6000	120	360

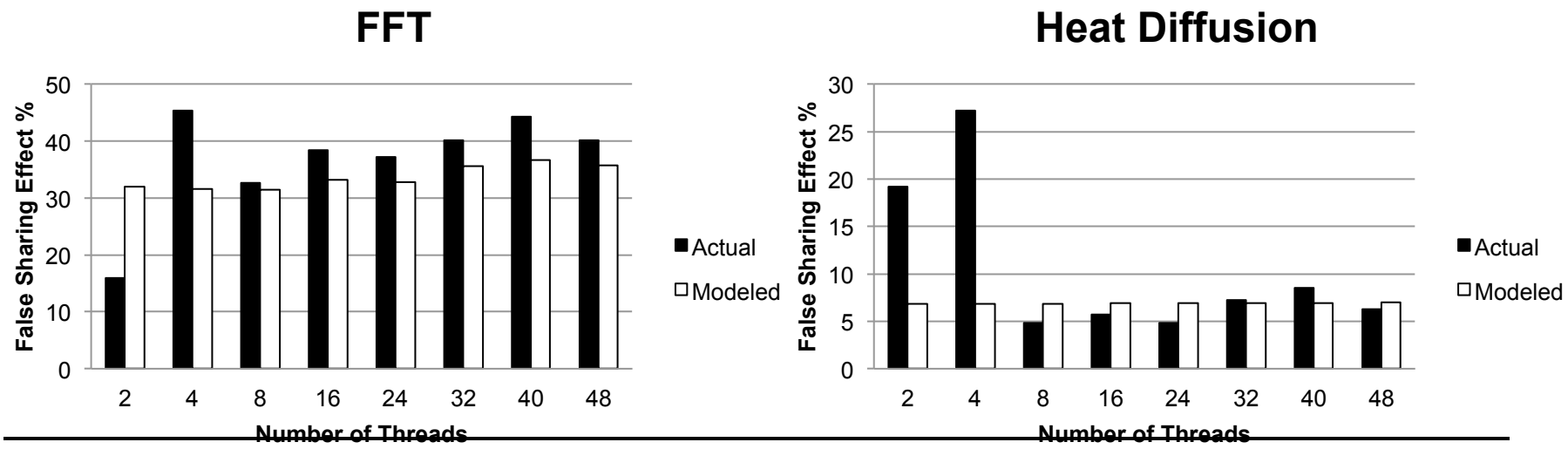


Modeling False Sharing at Compile-time

Compile-time assessment

- Analyze array references to generate a cache line ownership list
- Apply a stack distance analysis
- Compute the FS overhead cost

$$\frac{T_{fs\_measured} - T_{nfs\_measured}}{T_{fs\_measured}} \approx \frac{T_{fs\_modeled} - T_{nfs\_modeled}}{T^{*}_{fs\_modeled}}$$



M. Tolubaeva, Y. Yan and B. Chapman. Compile-Time Detection of False Sharing via Loop Cost Modeling. HIPS'12 Workshop in conjunction with IPDPS'12 (accepted)

Standard OpenMP Implementation

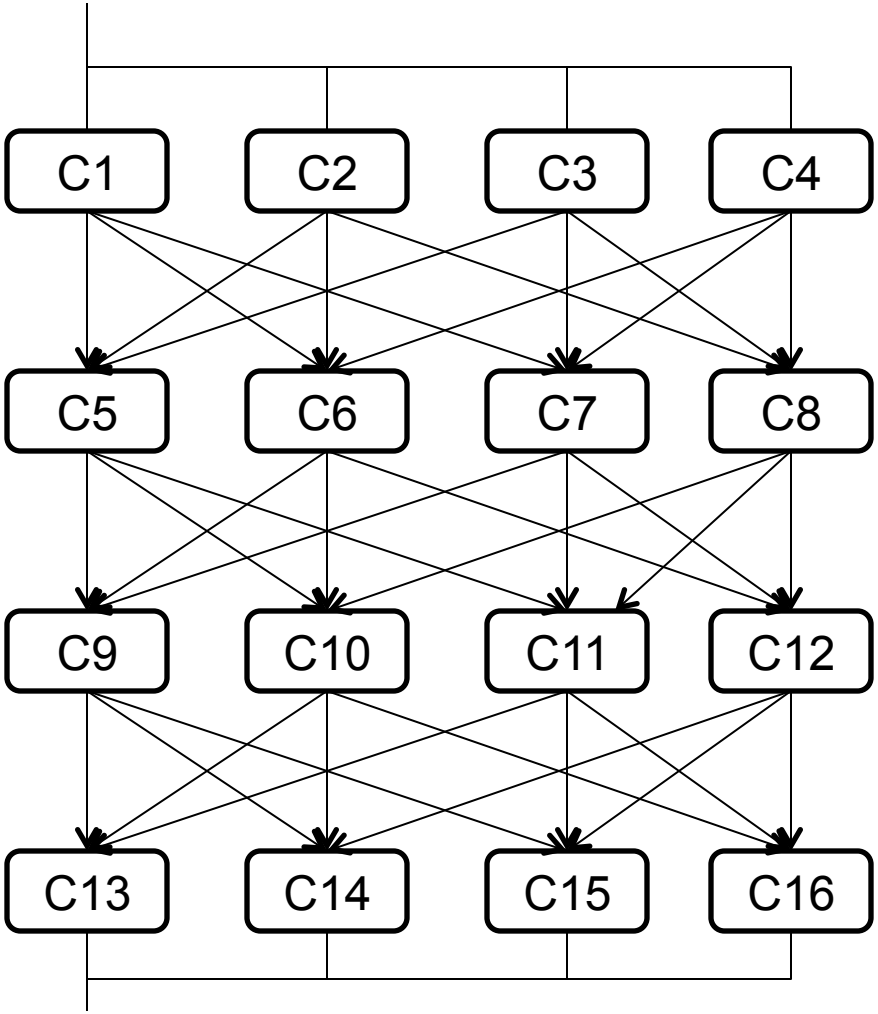
- Directives implemented via code modification and insertion of runtime library calls
  - Basic step is outlining of code in parallel region
  - Or generation of microtasks
- Runtime library responsible for managing threads
  - Scheduling loops
  - Scheduling tasks
  - Implementing synchronization
  - Collector API provides interface to give external tools state information
- Implementation effort is reasonable

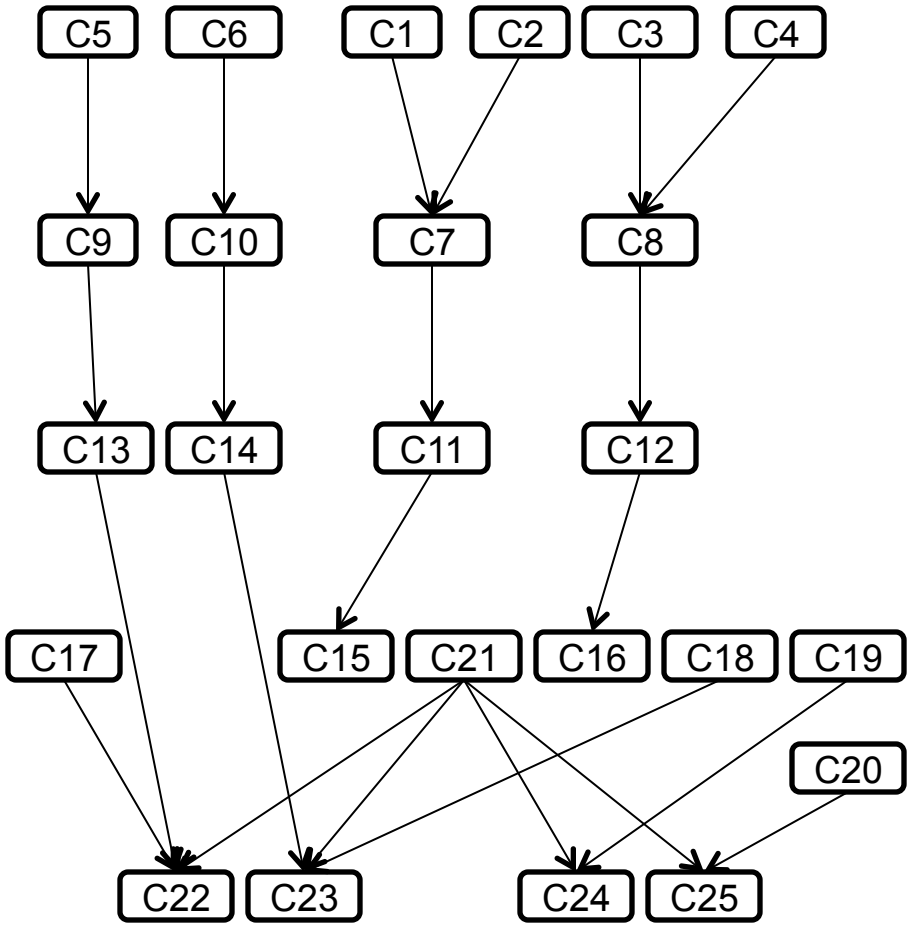
OpenMP Code	Translation

int main(void)
{
int a,b,c;
#pragma omp parallel \
private(c)
do_sth(a,b,c);
return 0;
}
	_INT32 main()
	{
	int a,b,c;
	/* microtask */
	void __ompregion_main1()
	{
	_INT32 __mplocal_c;
	/*shared variables are kept intact,
	substitute accesses to private
	variable*/
	do_sth(a, b, __mplocal_c);
	}
	...
	/*OpenMP runtime calls */
	__ompc_fork
	(&__ompregion_main1);
	...
	}

Each compiler has custom run-time support. Quality of the runtime system has major impact on performance.

Synchronization in OpenMP Execution





Heavy reliance on barriers for synchronization can lead to unnecessarily high overheads

Translation for Asynchronous Execution

- May be difficult for user to express computations in form of task graph
- Compiler translates “standard” OpenMP into collection of work units (tasks) and task graph
- Analyzes data usage per work unit
- Trade-off between load balance and co-mapping of work units that use same data
- What is “right” size of work unit?
  - Might need to be adjusted at run time

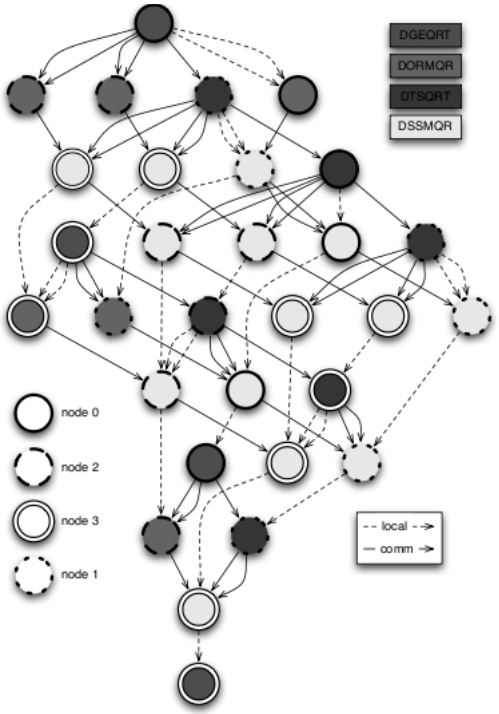


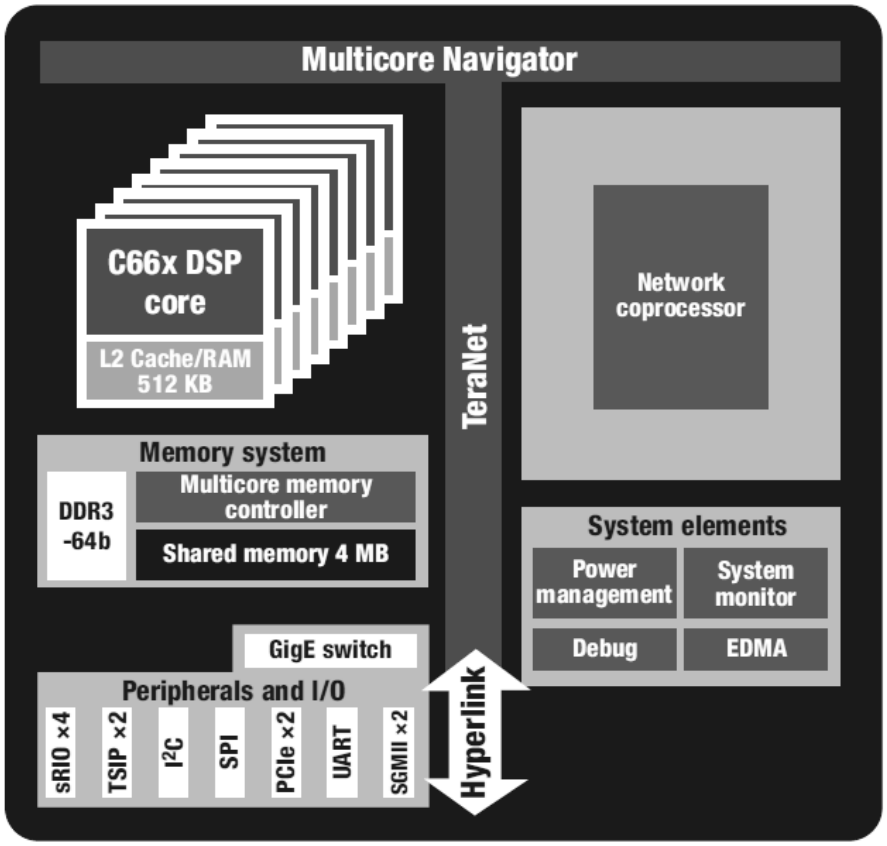
Fig. 5. DAG of QR for a 4x4 tile matrix.

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

Adding Machine Aware Translation

- Restructure work units
  - Merging or splitting work units for better granularity
  - Guided by parameterized cost model
- Application structural representation
  - Work units and dependences
  - Data distribution among places
- Compile time approximation
  - Data mapping onto places
  - Data binding with work unit
  - Decision honored by runtime
    - But may be adapted and refined.

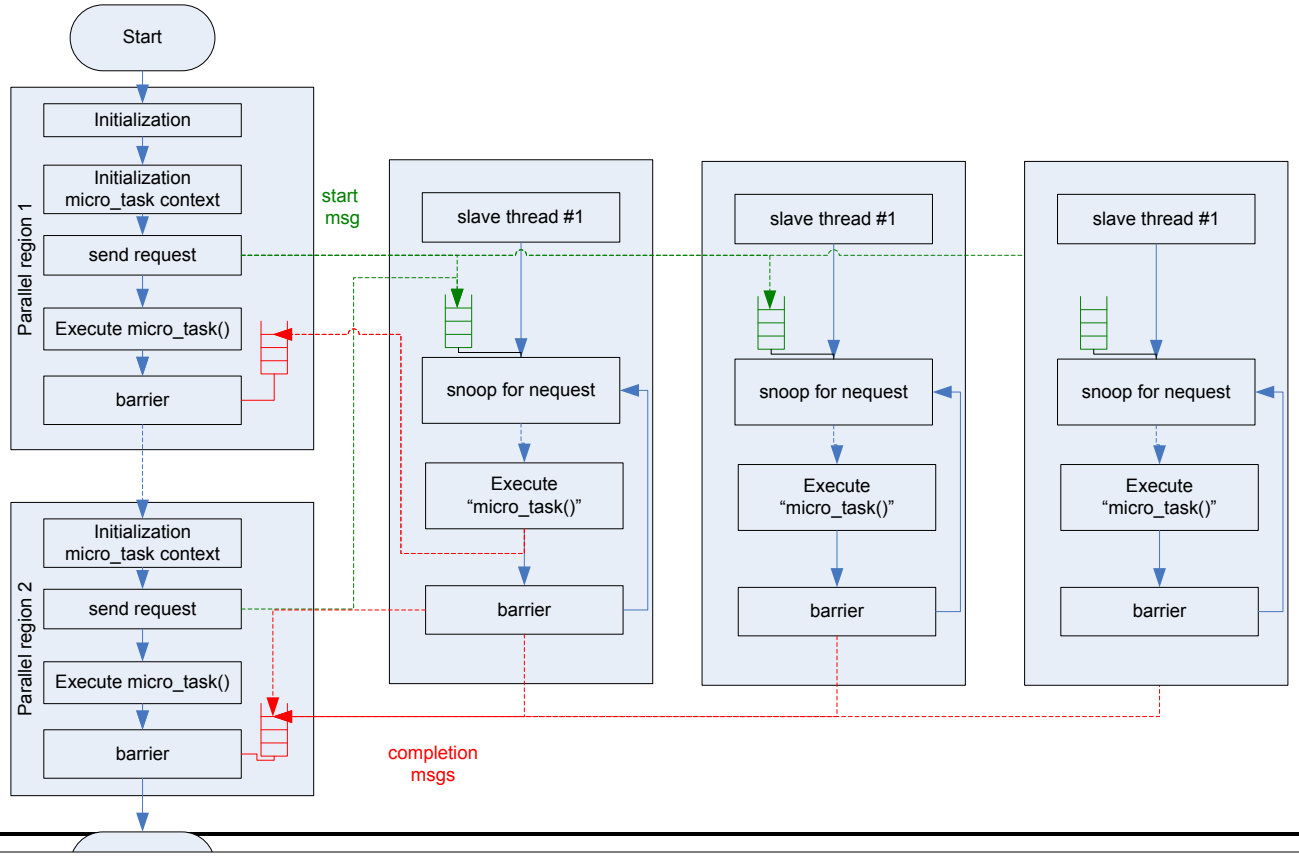
Consider Low-Power Architectures



- Other kinds of cores
- Different memory structure
  - Stack, caches, scratchpad
  - Virtual memory, Heaps
  - Segmented memory spaces
- TMDXEVM6678L EVM from TI
  - 8 DSP cores @ 1.25GHz
  - 32 KB L1D and L1P cache.
  - 512 KB L2 local cache.
  - 4 MB shared L2 cache.
  - 8 GB of shared external DDR3 memory at 12.8 GB/s.
  - Software-configurable cache
  - Slow access to DDR3

Adapting Translation to Best Exploit Memory

- Scratchpad memory, lack of coherent memory
- Slow shared memory, …



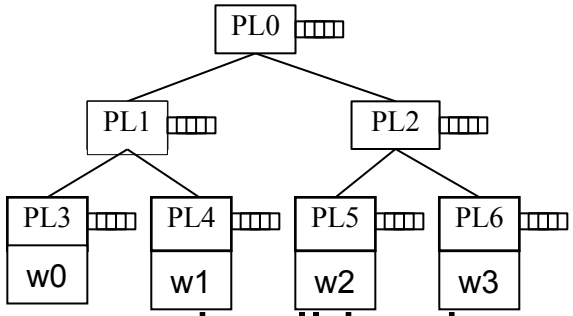
B. Chapman, L. Huang, E. Stotzer, E. Biscondi, A. Shrivastava, A. Gatherer. Implementing OpenMP on a High Performance Embedded Multicore MPSoC, pp 1-8, Proc. of Workshop on Multithreaded Architectures and Applications (MTAAP'09) In conjunction with International Parallel and Distributed Processing Symposium (IPDPS), 2009.

Agenda

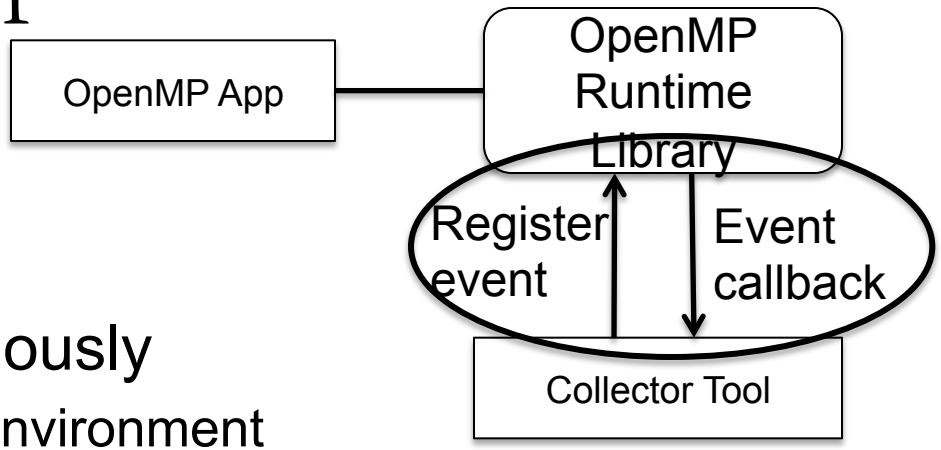
- Emerging HPC Architectures and their Programming Models
- OpenMP: An Evolutionary Approach to Node Programming
- Some Language Ideas for Locality
- Compiler Efforts: Increasing the Benefits
- Runtime and Tool Support

Runtime Locality-Aware Scheduling

- Locality-aware scheduling and data affinity
  - A worker executes tasks at ancestor places from bottom-up
  - Tasks from a place can be executed by all of the workers of the place subtree
- Lightweight synchronization
- Hybridization and heterogeneity
  - Helper thread(s)
  - Handling remote and async operations and call backs
- Runtime adaptation
  - Task-level auto-tuning



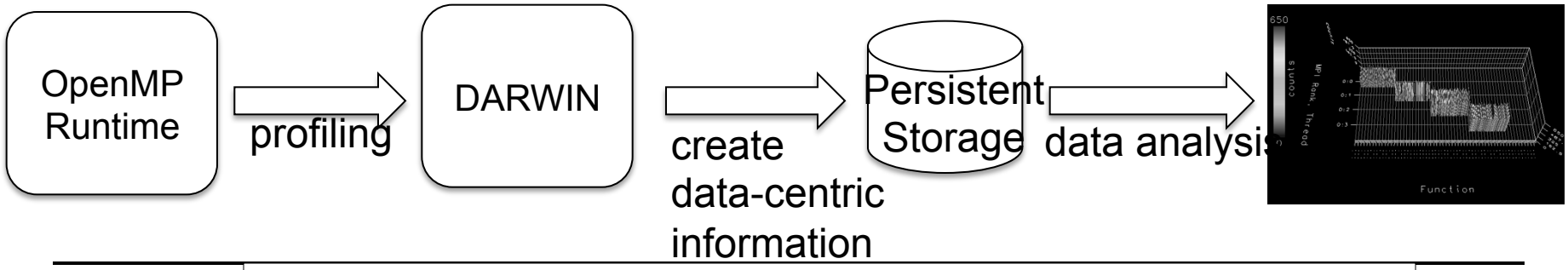
Runtime Must Adapt



- Runtime support to continuously
  - Adapt workload and data to environment
  - Respond to changes caused by application characteristics, power, (impending) faults, system noise
  - Provide feedback on application behavior
- Collector Interface, implemented in compiler's runtime, enables monitoring of OpenMP program
  - Enables tools to interact with OpenMP runtime library
  - Event based communication (OMP_EVENT_FORK, OMP_EVENT_JOIN,..)
- Do useful things based on notification

DARWIN: Feedback-Based Adaptation

- Dynamic Adaptive Runtime Infrastructure
  - Online and offline (compiler or tool) scenarios
  - Monitoring
    - Capture performance data for analysis via monitoring
    - Relate data to source code and data structures
    - Apply optimization and / or visualize
    - Demonstrated ability to optimize page placement on NUMA platform; results independent of numthreads, data size



Besar Wicaksono, Ramachandra C Nanjegowda, and Barbara Chapman. A Dynamic Optimization Framework for OpenMP. IWOMP 2011

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**

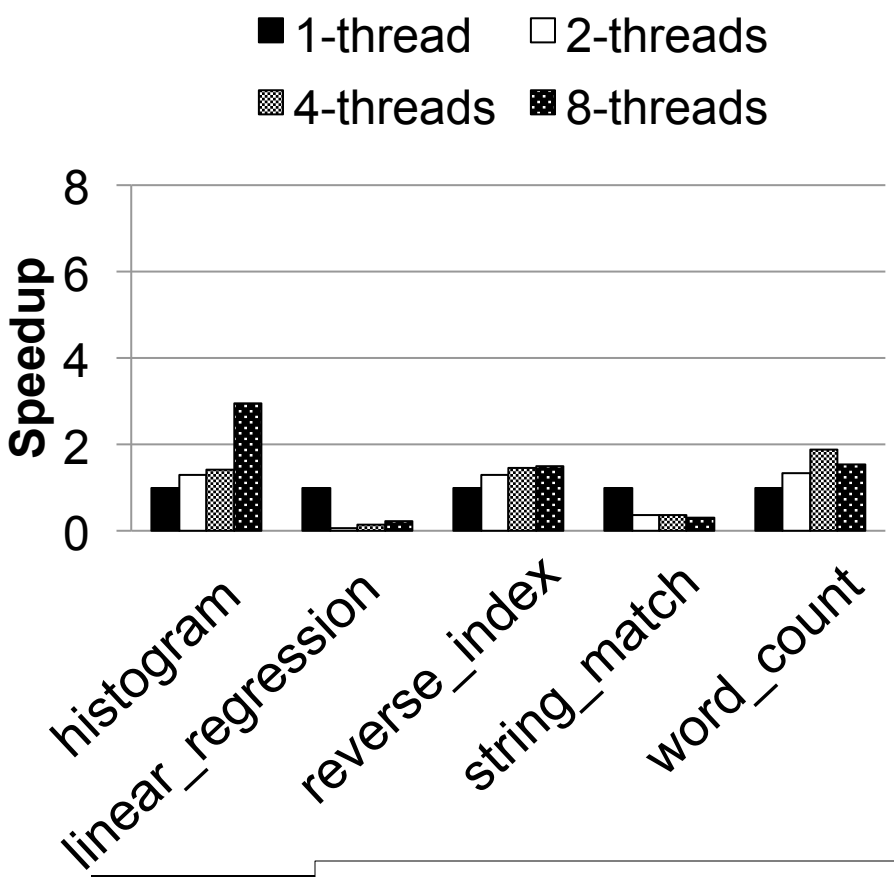
False Sharing: Data Analysis Results

- Determining the variables that cause misses

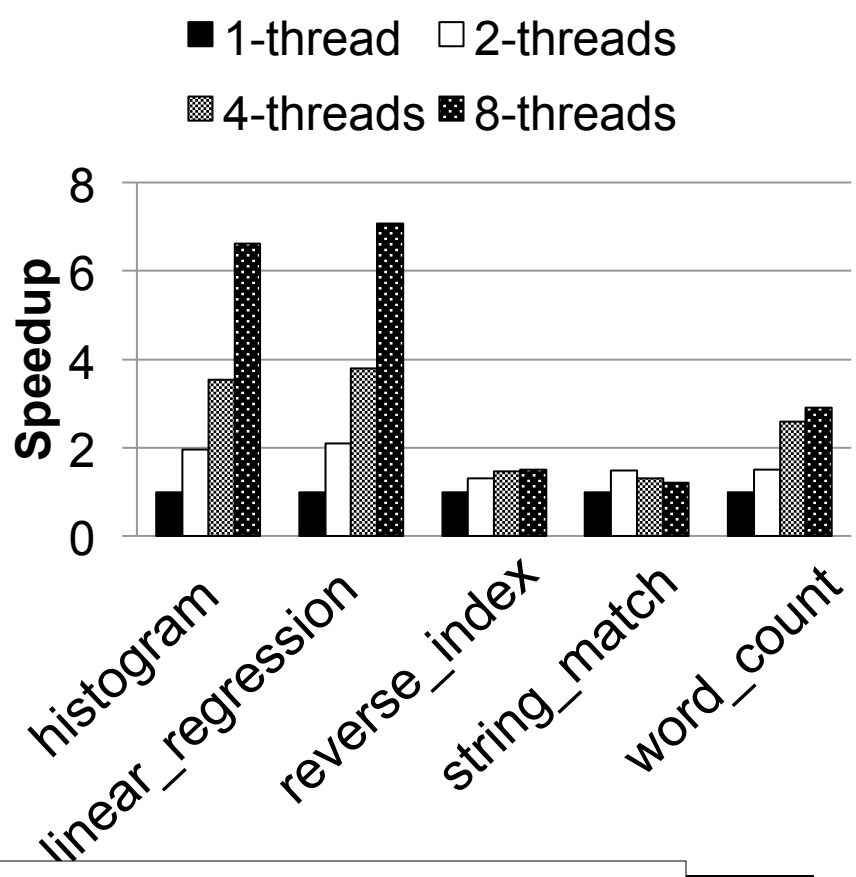
Program Name	**Global/static data**	**Dynamic data**
histogram	-	main_221
linear_regression	-	main_155
reverse_index	use_len	main_519
string_match	key2_final	string_match_map_266
word_count	length, use_len, words	-

Runtime Handling of False Sharing

Original Version



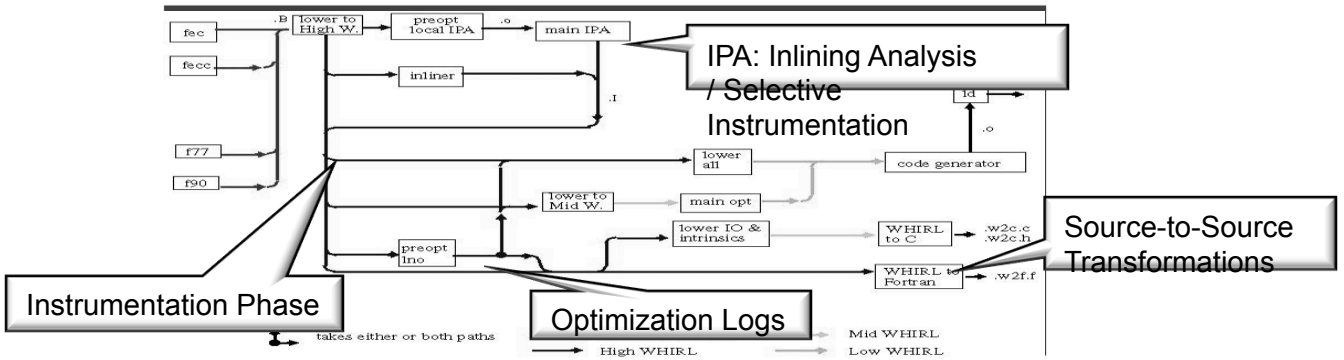
Optimized Version



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

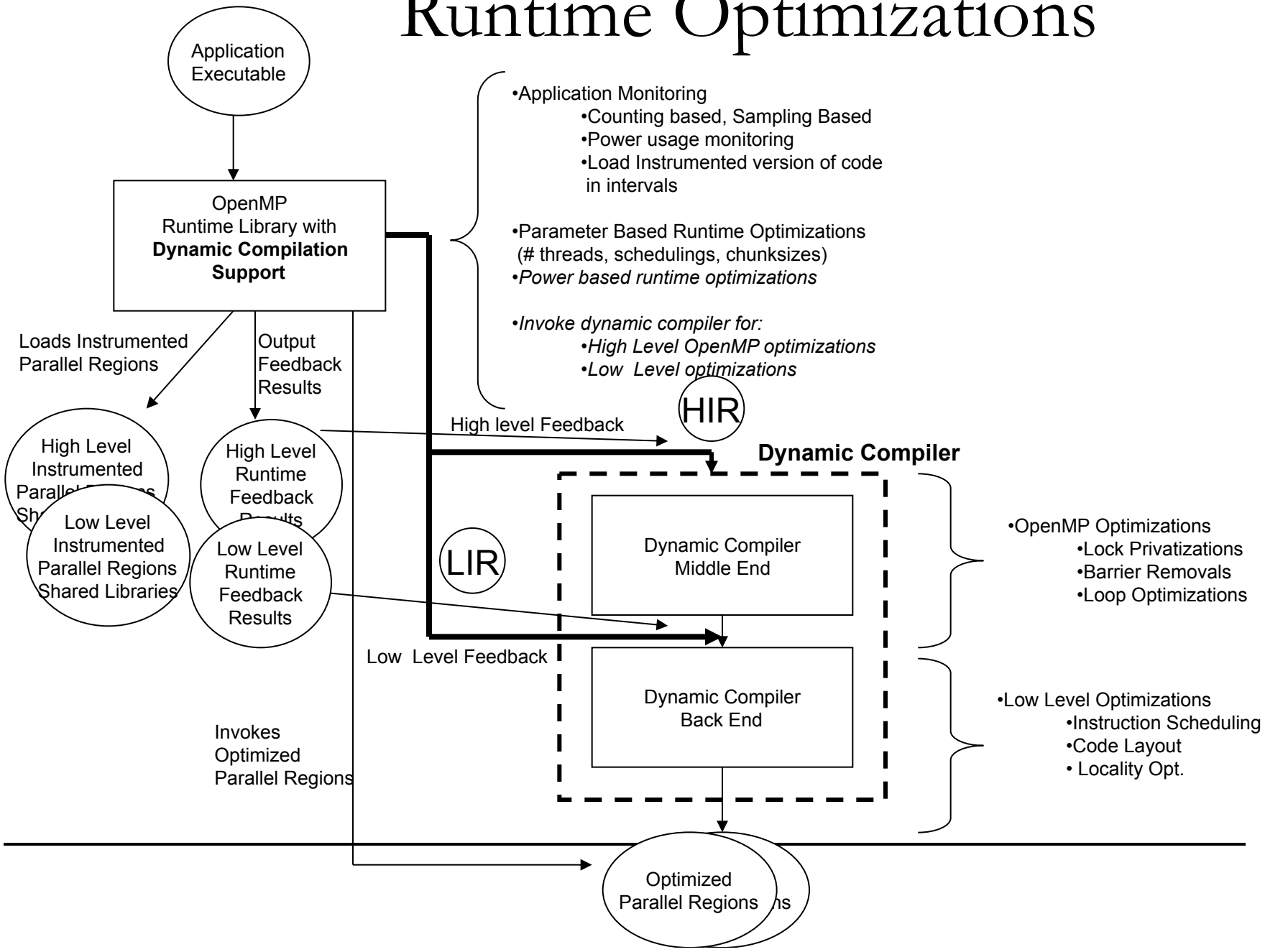
An Information-Rich Environment

- Compiler, tools collaborate to support application development and tuning
- All components cooperate to increase execution efficiency
- Coordinated management of system resources
- Application metadata used by compiler, tools and runtime
- Architectural information, system state, smart monitoring for adaptation on the fly
- Compiler modeling for dynamic optimization as well as feedback to user, tools



Oscar Hernandez, Haoqiang Jin, Barbara Chapman. Compiler Support for Efficient Instrumentation. In Parallel Computing: Architectures, Algorithms and Applications , C. Bischof, M. B¨ucker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.), NIC Series, Vol. 38, ISBN 978-3-9810843-4-4, pp. 661-668, 2007.

Runtime Optimizations



So What is Evolutionary?

- Hardware changes require us to rethink programming model, implementation, execution
  - Intra-node concurrency is fine-grained, heterogeneous
- Memory is scarce and power is expensive
  - Will need whole range of techniques to extract more concurrency, address locality and minimize power
- Not all the answers are in the programming model
  - Novel compiler translations
  - Extensive and powerful runtime to monitor and continuously adapt
  - Help evolutionary and revolutionary approaches alike