

REX: REthinking the “X” in the “MPI+X” for Exascale Nodes

Productive Programming Models for Exascale

Portland, OR, August 14-15, 2012

Yonghong Yan

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

University of Houston

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



About US

UH HPCTools Group

- Led by Barbara Chapman, member of OpenMP ARB
- 4 senior and ~18 graduate students (most Ph.D)
- <http://www.cs.uh.edu/~hpctools>

Major Research and Development

- OpenMP (NSF, DoE), OpenUH compiler
- PGAS: OpenSHMEM (DoD), CAF (TOTAL)
- *OpenACC test suite and compiler (NVIDIA)*
- Heterogeneous and Embedded related (TI, SRC, Freescale)

Myself, research assistant professor, OpenMP subcommittee member

- Use OpenMP, but more on implementing OpenMP and compilers



Background and Agenda

- Experience and thoughts on the node-level shared memory programming model for exascale
 - NSF HECURA project: eXtreme OpenMP: A Programming Model for Productive High End Computing
 - DoE pmodel project: Center for Programming Models for Scalable Parallel Computing
 - *Graduate students from compiler and parallel programming class*
- Motivation: Implicit vs Explicit
- REX Highlights
 - Abstract Machine Model, Compiler, Runtime
 - Some preliminary results



Programmers' Expectations for a programming model

- Will take care of her algorithms, but for me:
- Correct and deterministic performance
 - If she uses correctly, she wants to get expected performance
- Manageable control
 - Full control is good, but with complexity (MPI_Put/Get), ?
- Compiler, runtime and performance tools
 - HPC with Matlab, need a magic compiler



Inter-node

- All about communication, and domain decomposition
 - Library approach do the work: MPI, GA, and OpenSHMEM
 - Realize the SPMD model on cluster
- MPI
 - Well balanced between programmability and performance
 - Legacy codes
- Language approach (CAF and UPC)
 - Some become implicit → Less/lose control
- Chapel and X10 – compiler/runtime



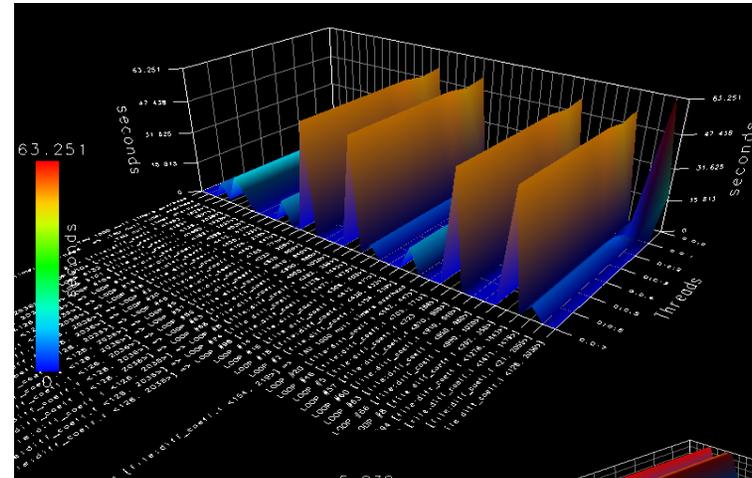
Talking about shared memory programming

- Access data anywhere, and sometimes implies *uniformly*
 - NUMA/NUCA effects
- Data consistency taken care by cache-coherence
 - False-sharing
- Barrier causes threads to sync
 - Barriers are global and heavy
- When using locks, be careful of deadlock
 - But it is your fault



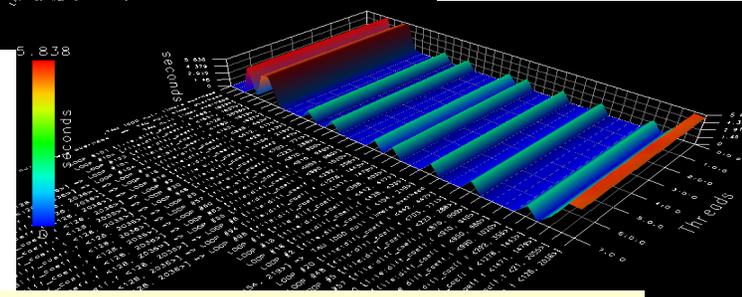
Small “Mistakes”, Big Consequences

- 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

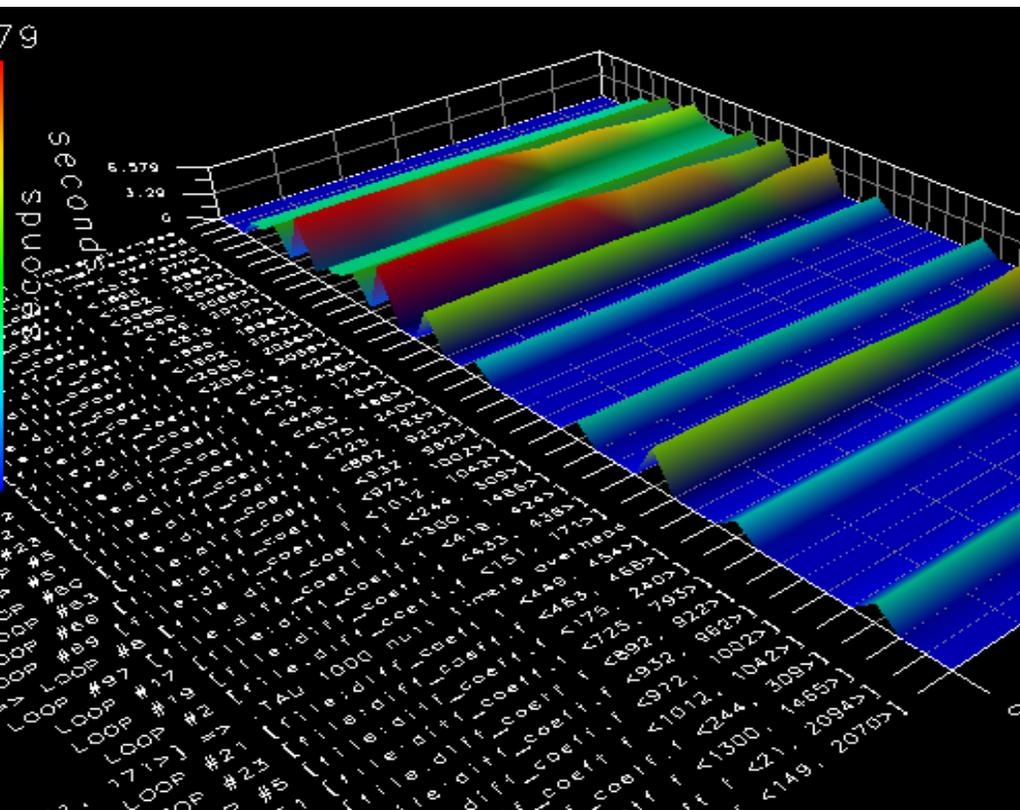


In the OpenMP version, a single procedure is responsible for 20% of the total time and is 9 times slower than the MPI version . Its loops are up to 27 times slower in OpenMP than MPI.



A Solution: Privatization

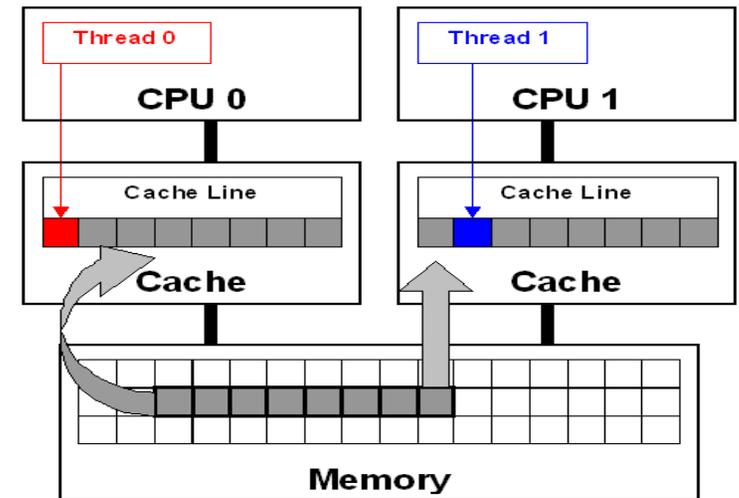
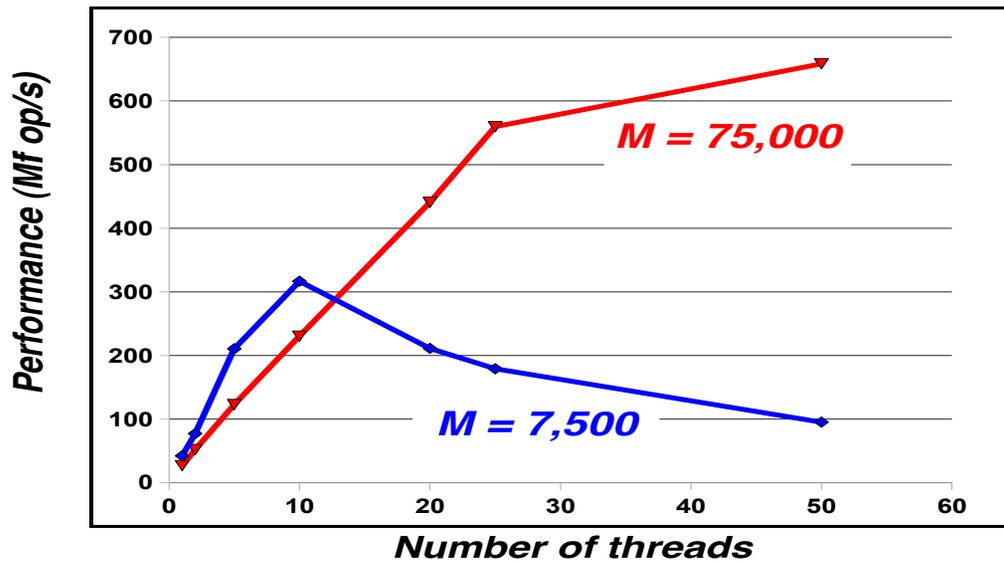
OpenMP Optimized Version



- 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.

False-sharing at Work

```
!$omp parallel do default(shared) private(i) &  
!$omp schedule(static)  
  do i = 1, m  
    x(i,j,k) = x(i,j,k-1) + x(i,j-1,k)*scale  
  end do  
!$omp end parallel do
```



For a higher value of M , the program scales better

Intra-node

- Correct performance is not always good
 - Productivity is not always good
- Much research in architecture
 - Compiler, debugger, JIT, JITMA and JITMA and JITMA
 - Selected below
 - Compiler
 - Selected below



Disciplined Shared Memory Programming

- Engineer or formulate performance tuning *practices*
 - Currently live in the mind
- A programming model with explicit and full control vs implicit
 - *Why not we make more explicit at the beginning*
 - It is a tradeoff on what to be explicit/implicit

PACT 2011 Best Paper: DeNovo: Rethinking the Memory Hierarchy for Disciplined Parallelism, Byn Choi (UIUC), etc



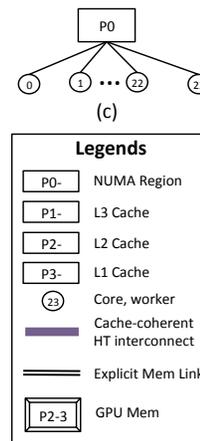
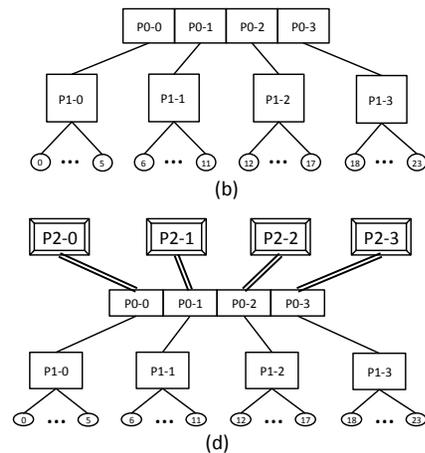
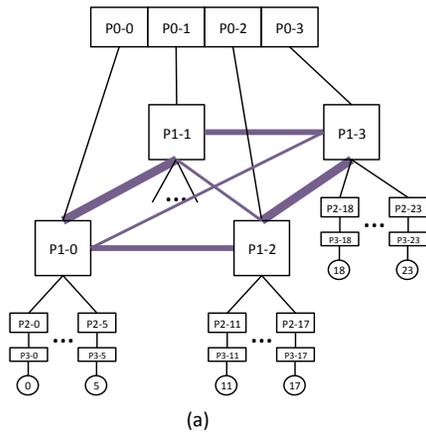
REX Highlights

- To provide to programmers a manageable control of node architectures through an *abstract machine model*
- *Explicit locality and analyzable data operations* to enable aggressive machine-aware compilation and runtime adaptation
- Abstract Machine Model (AMM)
 - Hierarchical Place Trees (HPT)
- Explicit Locality and Data Operations
 - Analyzable by compilers
- Compiler cost model and machine aware optimizations
 - Static mapping, and data placement
- Runtime
 - Dynamic refinement and adaptation



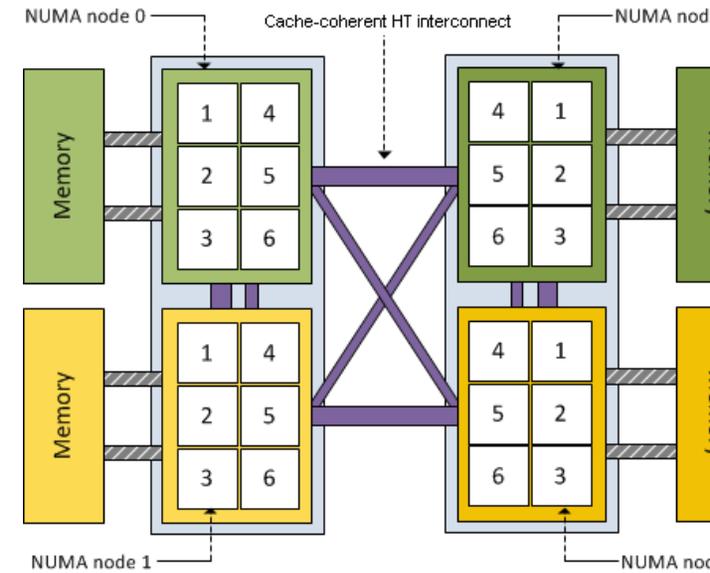
REX AMM: Hierarchical Place Trees

- Place denotes storage, and links as edges
 - Cache, SDRAM, device memory
- PE as worker threads
- Different HPT deployments
 - Trade-off between locality and load-balance



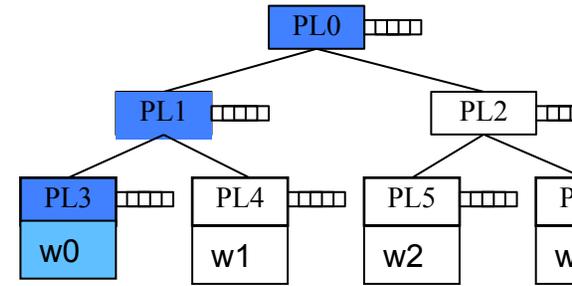
Legends

- P0- NUMA Region
- P1- L3 Cache
- P2- L2 Cache
- P3- L1 Cache
- 23 Core, worker
- Cache-coherent HT interconnect
- Explicit Mem Link
- P2-3 GPU Mem



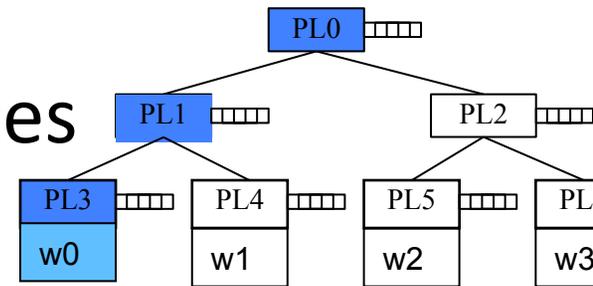
How HPT works (1)

- Data in a child place are subset of its parent
 - May not be coherent
 - Optimization should reduce movement upwardly
- Data are placed at specific place by
 - Programmers, static placement (compiler) and at runtime
 - If undecided, data can always placed at the root
- Work unit (wunit), parallel region, tasks, etc are created with a target place (implicit or explicit)
 - Assume to process data residing on that place

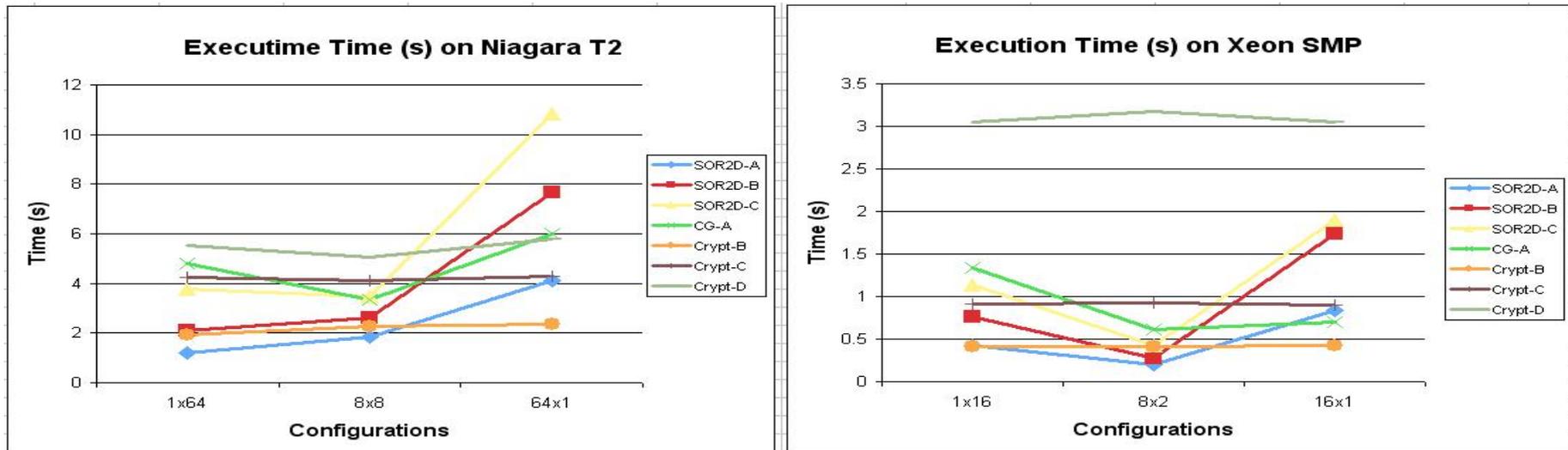


How HPT works (2)

- A worker executes tasks of ancestor places
 - w0 executes tasks from pl3, pl1, pl0
- Tasks from a place can be executed by all of the workers of the place subtree
 - Affinity in scheduling
 - If Tpl1 is scheduled by w2 or w3, data has to be transferred to the top level (pl0) and then down to either pl5 or pl6



HPT preliminary results

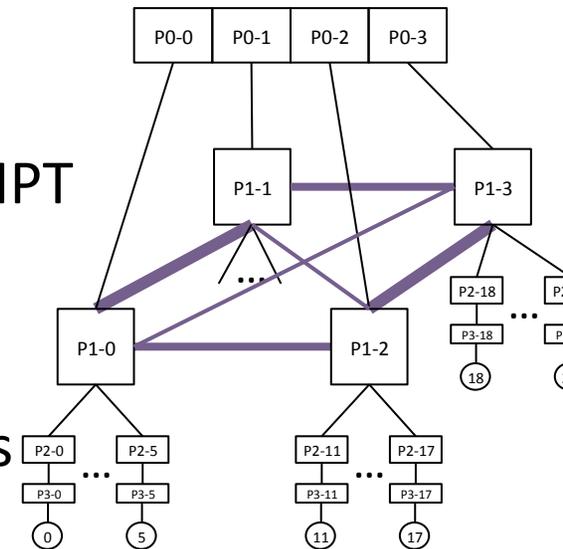


- 8x2 and 8x8 configs expect better performance
 - True for SOR2D, A, B, C and CG A on Xeon SMP
 - True for SOR2D C, CG A, and B, and Crypt C and D on T2
- Better performance for Locality-sensitivity apps
 - Deployment to be consistent with memory architecture

Hierarchical Place Trees: A Portable Abstraction for Task Parallelism and Data Movement, Yonghong Yan, Jisheng Zhao, Yi Guo and Vivek Sarkar, LCPC 2010

The HPT's Role

- Programmers:
 - A portable representation of memory hierarchy as HPT
 - Explicit data and work unit binding
- Compilers: machine aware compilation
 - Static data placement and binding of wunit to places
 - Cost model guided
- Runtime: adaptivity and refinement of static decision
 - Locality-aware scheduling
 - Cost-guided data/task migration
 - Helper threads



Programming Interface

- Explicit locality and data operations
 - `rex_malloc(size_t, place_t)`
 - `rex_[async]memcpy(void * dst, void * src, size_t, place_t dst_pl, place_t src_pl)`
 - `rex_[async]memset`
 - `rex_mmap` and `rem_msync`
 - `#pragma rex parallel [at<pl> | atpof<ptr>] [team ...]`
 - `#pragma rex task [at<pl> | atpof <ptr>] [team ...]`
- All compiler recognizable for optimizations



Programming Interface

- Parallel Patterns
 - Worksharing, async tasks, data/event driven, data movement operations as schedulable work unit
- Synchronizations
 - `#pragma rex barrier`: only apply to the enclosing parallel region, not global



Realistic on compilers

- Most effective compiler optimizations: Vectorization
 - Cannot play magic
- Compiler needs more information
 - Microarchitectures → micro-opt
 - Machine architectures → macro-opt
- The lack of macro compilation
 - Need language support



REX Compiler

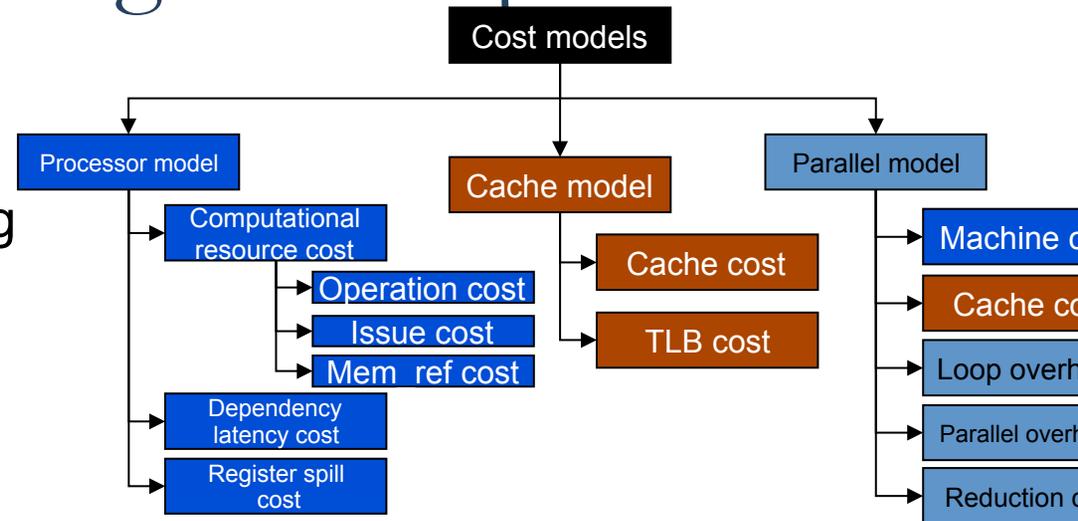
- High-level parallelism aware compilation
 - Frontend, unlimited resources for idea execution
- Machine aware compilation based on HPT
 - Static mapping and data placement on to HPT
- Cost model to guide the two compilations
 - Simulation and search
 - Database for runtime refinement and adaptation



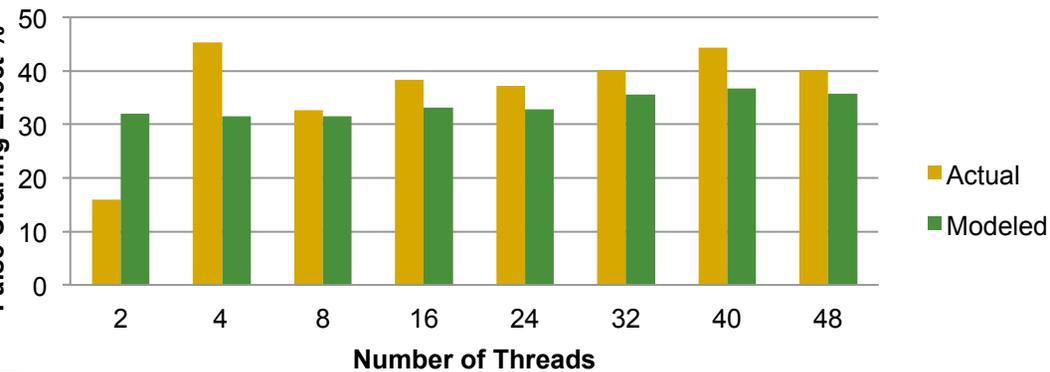
Modeling False Sharing at Compile-time

- Cache access simulation
- Cost estimation of memory access taking into account of shared cache

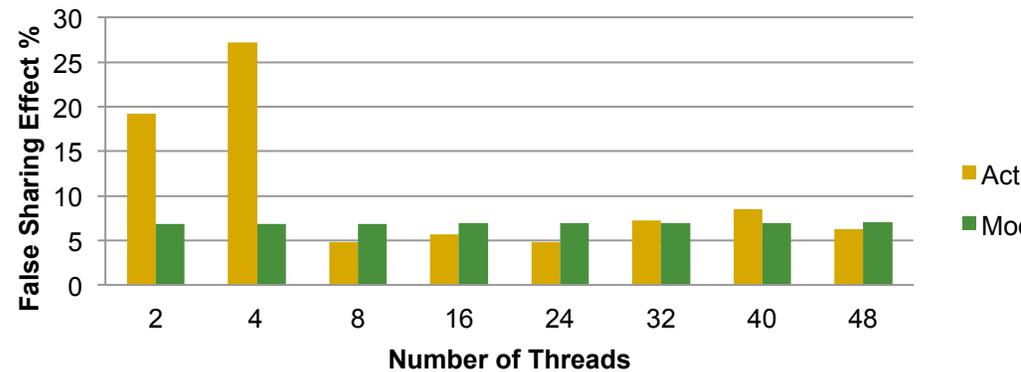
$$\frac{T_{fs_measured} - T_{nfs_measured}}{T_{fs_measured}} \approx \frac{T_{fs_modeled} - T_{nfs_modeled}}{T_{fs_modeled}^*}$$



FFT



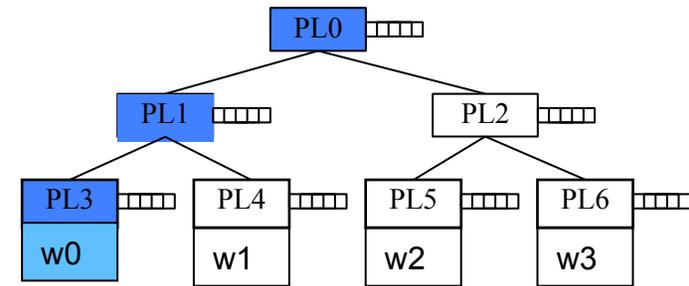
Heat Diffusion



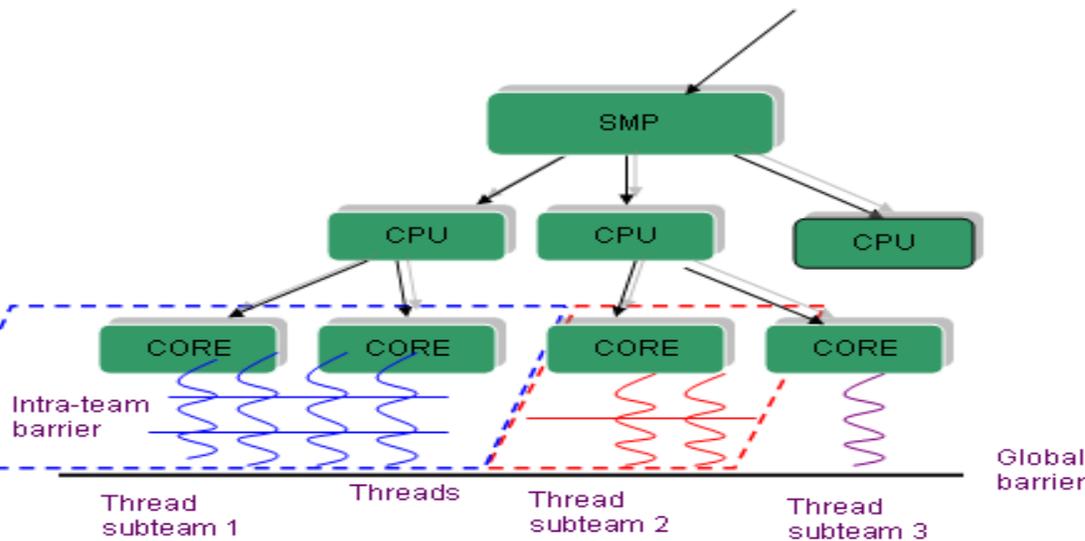
M. Tolubaeva, Y. Yan and B. Chapman. Compile-Time Detection of False Sharing via Loop Cost Modeling. HIPS'12 Workshop in conjunction with IPDPS'12

REX Runtime

- Locality-aware scheduling and data affinity with HPT
- Lightweight and localized synchronization
- Hybridization
 - Handling remote and async operations
- Runtime adaptation
 - Task-level auto-tuning
 - Helper thread for dynamic compilation



Subteams of Threads?



```
(j=0; j< ProcessingNum; j++)
```

```
#pragma omp for schedule(dynamic)
```

```
subteam(2:omp_get_num_threads()-1)
```

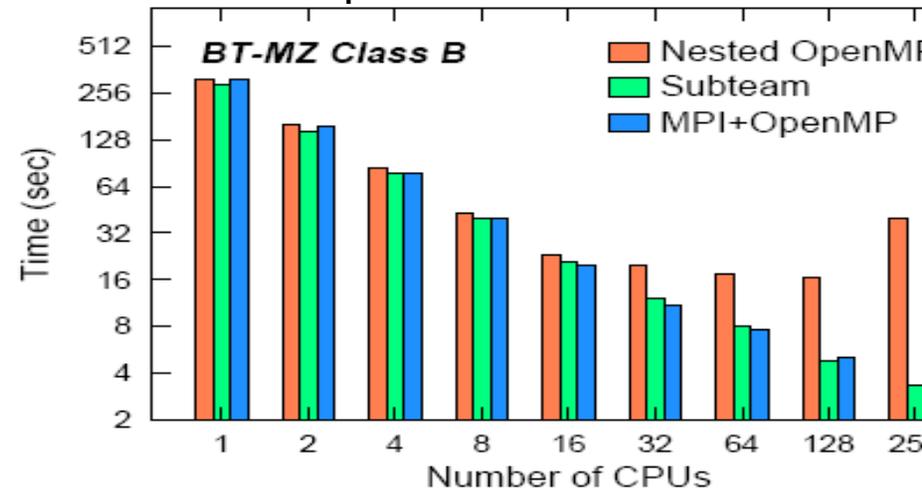
```
for (k=0; k<M; k++) {
```

```
    ProcessData(); // data processing
```

```
} // subteam-internal barrier
```

Thread Subteam: finer grained resource management

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



Increases expressivity of single-level parallelism

Conclusion

- Programming model vs lib/languages
 - Model not changed, lib/lang changed
- Some ideas are inspired from PGAS/APGAS languages
- Revolutionary in compiler/runtime/tools
 - Evolutionary in programming interfaces
- HPC education for new generation graduate



