Enabling Exascale Programming: A System-Wide Challenge

Barbara Chapman

University of Houston

31e Forum ORAP
Paris, 3/25/2013

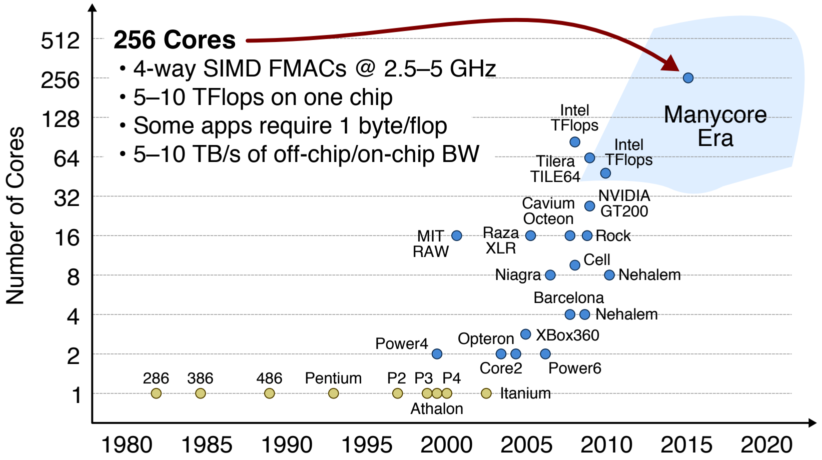
Acknowledgements: NSF CNS-0833201, CCF-0917285; DOE DE-FC02-06ER25759, TOTAL SA

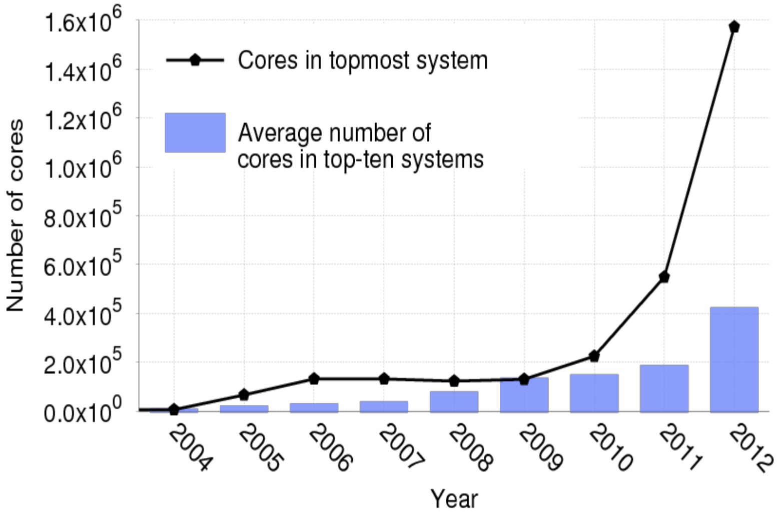
http://www.cs.uh.edu/~hpctools

Agenda

- **The Challenge**
- Programming Models: What’s Out There?
- Compiler Efforts: Increasing the Benefits
- Runtime Support and System Interactions

Exascale: Anticipated Architectural Changes





- Massive (ca. 4X) increase in concurrency
  - **Mostly within compute node**
- Node architecture is changing considerably
  - Core count, heterogeneity, memory size & BW, power, resilience
- Balance between compute power and memory changes significantly
  - 50x FLOPs of 20PF HW but just a small increase in memory
  - Memory access time lags further behind

Multicore Programmer’s Wish List

- Rewriting applications from scratch requires considerable time and effort
  - Need easy way to parallelize existing codes
  - Incremental migration path essential for major applications
- …with familiar and/or commodity programming models
  - Not all programming models are created equal
  - None are perfect, but industry adoption is critical

HPC Programming Model Ingredients

- Performance
  - Parallelism; load balance; minimization of waits
- Portability
  - Across diverse, perhaps heterogeneous systems
- Power-saving
  - Mainly via locality

Multiple layers of potentially different kinds of parallelism in hardware

And let's not forget Productivity

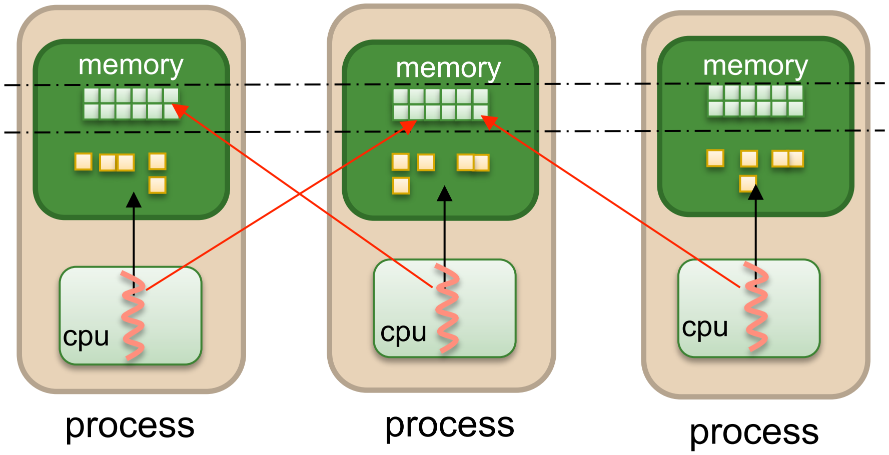
Agenda

- The Challenge
- **Programming Models: What's Out There?**
- Compiler Efforts: Increasing the Benefits
- Runtime Support and System Interactions

The PGAS Approach

Characteristics:

- Global view of data
- Data affinity part of memory model
- Single-sided remote access



Advantages:

- more representative of modern NUMA architectures
- Works well for distributed interconnects with RMA support
- Productivity and performance

Language Extensions:
UPC, Coarray Fortran (CAF), Titanium

Libraries:
OpenSHMEM, Global Arrays, GASPI

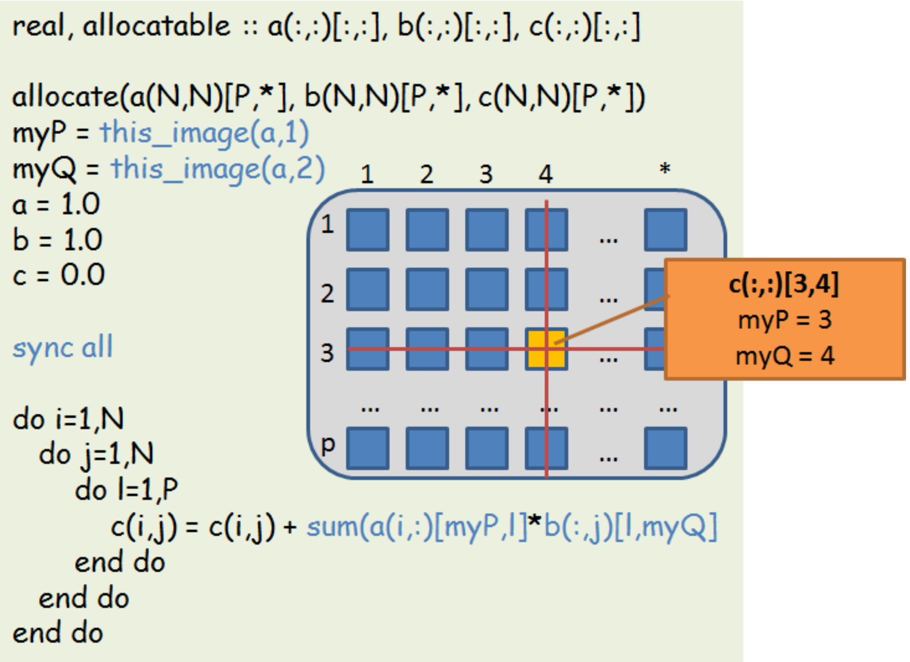
“APGAS”:
X10, Chapel, Fortress, CAF 2.0

Example: Coarray Fortran (CAF)

- SPMD execution model; each executing unit is called an **image**
- Remotely accessible data declared as **coarrays**
- Adds support for various synchronization mechanisms to the language (e.g. barriers, locks)
- Support for teams, collectives, atomics, and event-based synchronization around the corner
- **Part of the Fortran standard (Fortran 2008)**

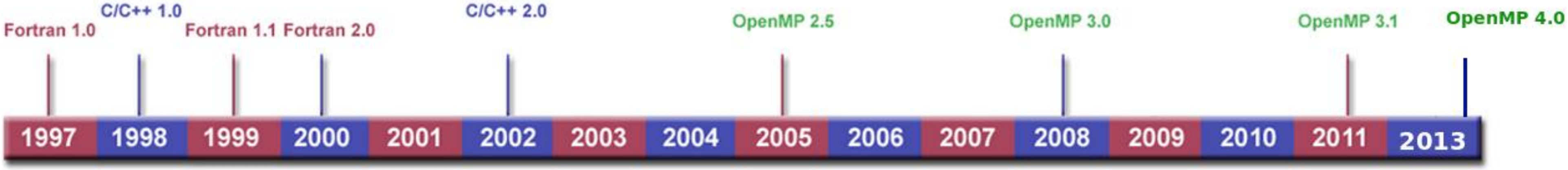
Matrix Multiply example:

- Coarrays allocated symmetrically
- ilage index info obtained through intrinsic function calls
- **sync all** for global barrier
- Remote data accesses are achieved through co-indexed coarray references



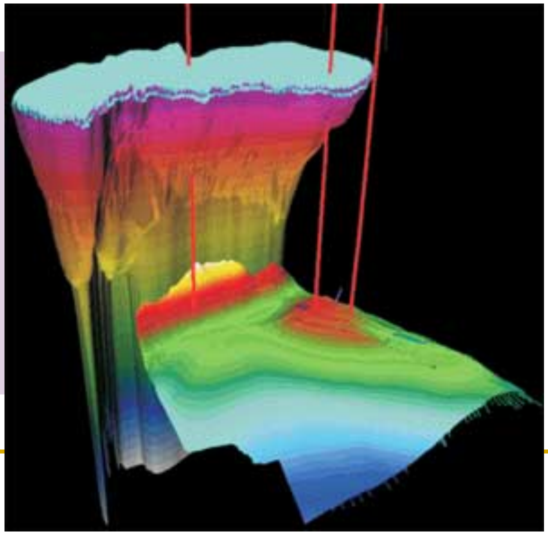
```
real, allocatable :: a(:,:)[:,:], b(:,:)[:,:], c(:,:)[:,:]

allocate(a(N,N)[P,*], b(N,N)[P,*], c(N,N)[P,*])
myP = this_image(a,1)
myQ = this_image(a,2)
a = 1.0
b = 1.0
c = 0.0

sync all

do i=1,N
  do j=1,N
     do l=1,P
        c(i,j) = c(i,j) + sum(a(i,:)[myP,l]*b(:,j)[l,myQ])
     end do
  end do
end do
```

Defacto Mature Standard - OpenMP

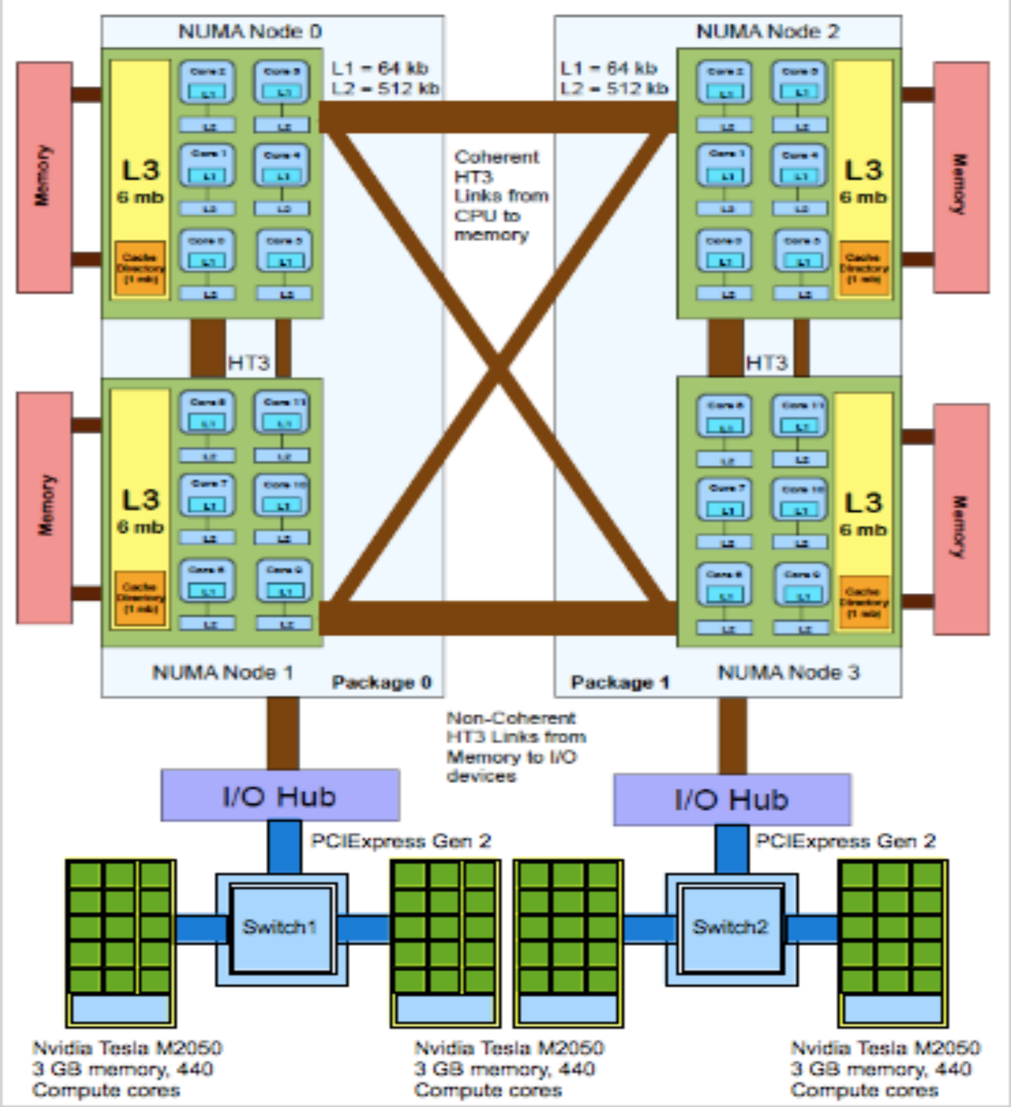


- High-level API for shared memory programming
  - Widespread vendor support and a large user base
  - User makes strategic decisions; compiler figures out details
- OpenMP code is portable
  - Across compilers, runtimes
  - Mainstream compilers for Fortran, C and C++ support OpenMP

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



Keeping OpenMP Relevant



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

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

OpenMP ARB 2013

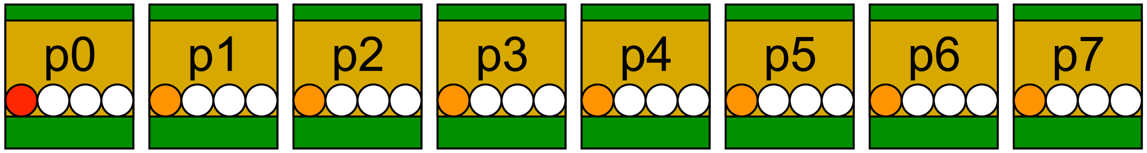
Upcoming OpenMP 4.0

- Release Candidate 1 @ SC12
- Release Candidate 2 ~March 2013
- Candidate topics:
  - Accelerator
  - Affinity and locality
  - Task extensions: task group and dependent tasks
  - Error model
  - SIMD extensions
  - Tools interface
  - User-defined reductions

OpenMP 4.0 Affinity Proposal

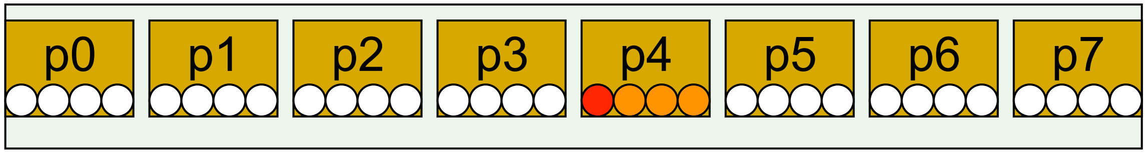
- OpenMP places and thread affinity policies
  - `OMP_PLACES` to describe places in system
  - `affinity(spread|compact|true|false)`
- `SPREAD`: spread threads evenly among the places

`spread 8`

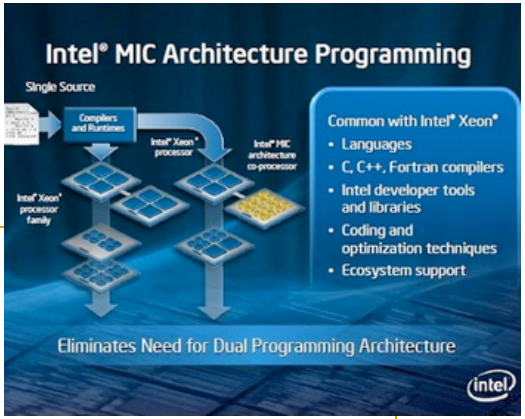


- `COMPACT`: collocate OpenMP thread with master thread

`compact 4`



OpenMP SIMDization



Intel® MIC Architecture Programming

Common with Intel® Xeon®
- Languages
- C, C++, Fortran compilers
- Intel developer tools and libraries
- Coding and optimization techniques
- Ecosystem support

Eliminates Need for Dual Programming Architecture

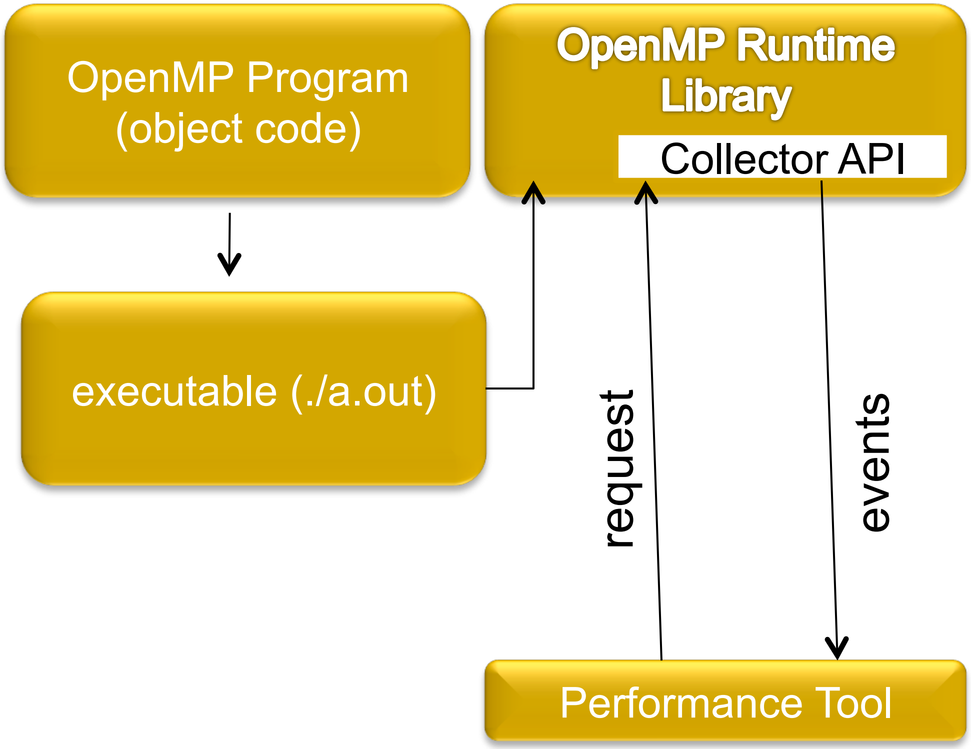
Single Source
Compilers and Runtimes
Intel® Xeon® processor
Intel® MIC architecture co-processor
Intel® Xeon® processor family
intel

```
#pragma omp simd [clause[[,] clause] ...] new-line
for-loops
where clause is one of the following:
safelen( length ) linear( list[:linear-step] ) aligned( list[:alignment] )
private( list ) lastprivate( list ) reduction( operator:list ) collapse( n )
```

- The simd construct can be applied to a loop
- Indicates that it can be transformed into a SIMD loop
- Multiple iterations of the loop can be executed concurrently using SIMD instructions
- Can also be combined with parallel loop (for or do)

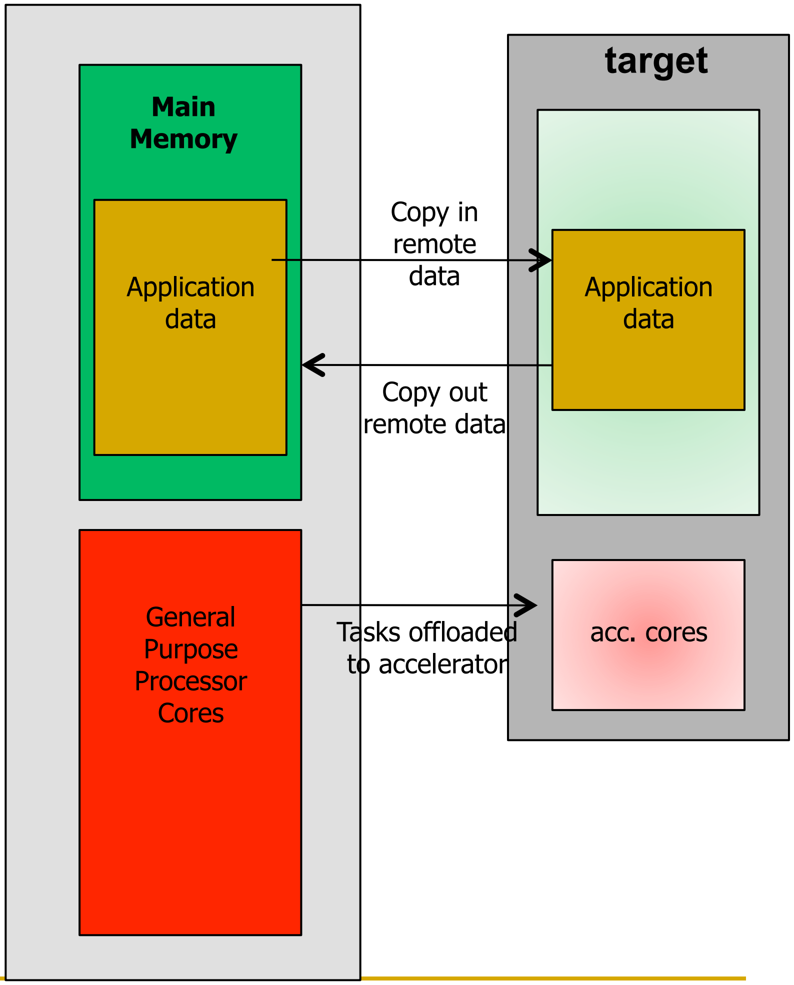
OpenMP Performance Tools Interface

- int__omp_collector_api(void *msg)
- Single routine used by tools to communicate with runtime.
- One call, many requests.
- Supports events/states needed for statistical profiling and tracing tools
- Current work extends original design from Sun



OpenMP 4.0 Accelerator Model

- Execution Model: Offload data and code to accelerator
  - *target* construct creates tasks to be executed by devices
- Memory Model:
  - shared data copies synchronized implicitly at end of target construct regions or explicitly using a target flush construct.
- Intended to work with wide variety of accelerators
- User maps data to and from the device memory
- Enables hierarchical parallelism



OpenACC

- High-level directive-based programming model for heterogeneous architectures
  - Collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran
  - Enables scientific Fortran and C programmers to take advantage of heterogeneous CPU/GPU computing systems.
  - Takes multiple levels of parallelism for GPUs into account

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

OpenACC 2.0

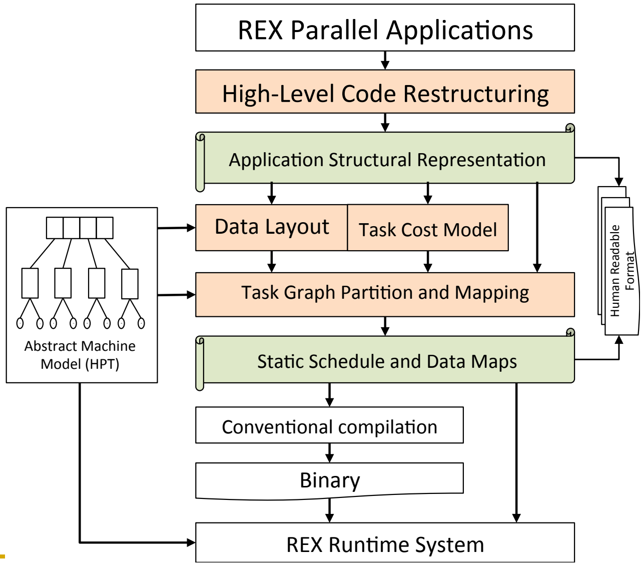
- Support for procedure calls
  - Compiler needs to know what procedures are needed on the device – This is done by the ‘routine’ directive
  - Routine with a ‘bind’ clause tells the compiler to call a routine with a different name when calling on the accelerator
- Nested Parallelism
- Device Specific Tuning
  - Tuning for multiple devices in a single program
- Data Management Features
  - To manage data lifetimes on the device
- Async Clause
  - To resolve dependencies between multiple async handles without requiring the host thread to wait
- Loop directive additions
- New API routines

Agenda

- The Challenge
- Programming Models: What’s Out There?
- **Compiler Efforts: Increasing the Benefits**
- Runtime Support and System Interactions

Machine Aware Compilation

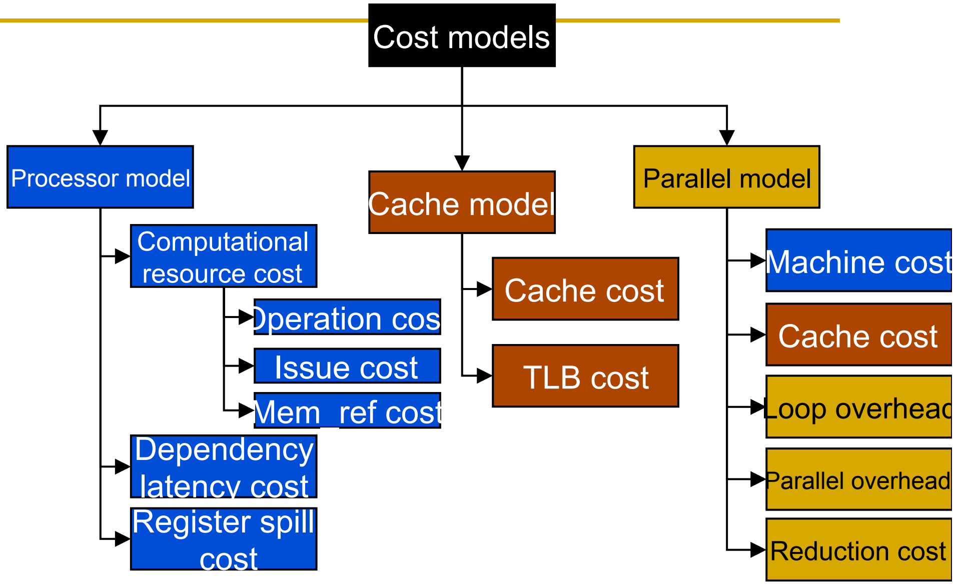
- Restructure work units
  - Merging or splitting work units for better granularity
  - Guided by parameterized cost model
- Application structural representation
  - Work units and dependences
  - Data distribution among places
- Compile time approximation
  - Data mapping onto places
  - Data binding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

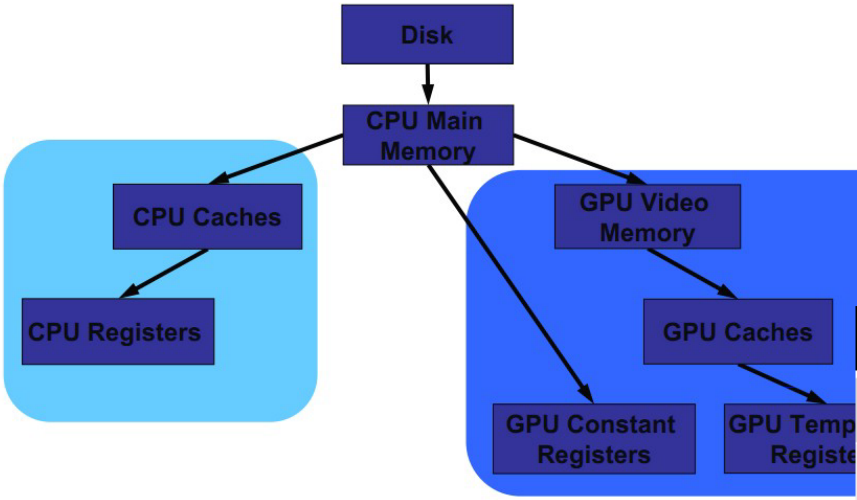

REX Parallel Applications
High-Level Code Restructuring
Application Structural Representation
Data Layout
Task Cost Model
Task Graph Partition and Mapping
Static Schedule and Data Maps
Conventional compilation
Binary
REX Runtime System
Abstract Machine Model (HPT)
Human Readable Format

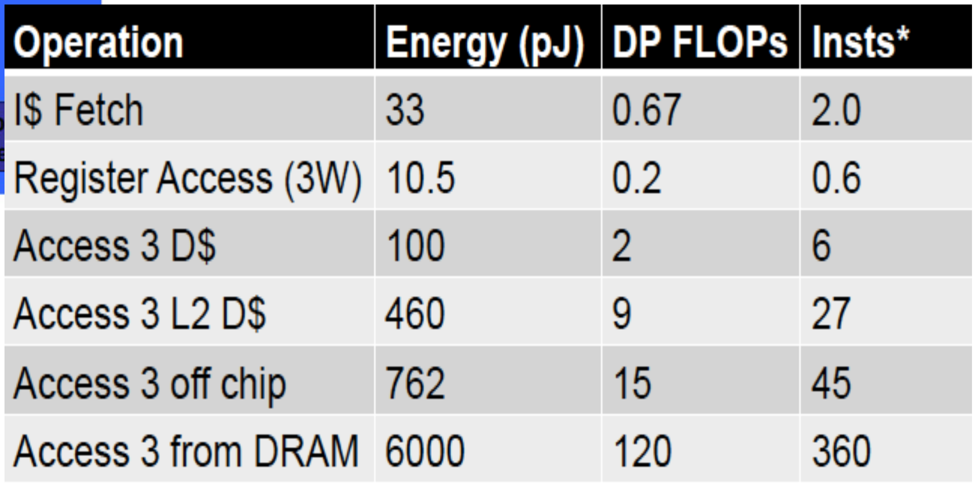
Compiler Cost Models Guide Translation

- Conventional cost model
  - Mostly evaluates cache effects of uniprocessors
- Modeling 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
- Node model
  - Multiple kinds of cores, interconnect, structure of memory hierarchies
- Supports compile-time and runtime optimization
  - Data placement and affinity between tasks and data
  - Mapping task graphs to the hardware architectures
  - Guided energy-aware scheduling

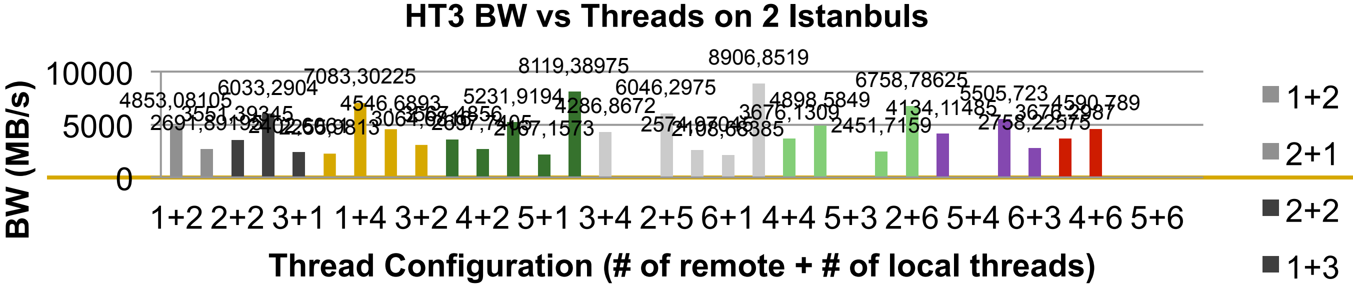
What to Model?



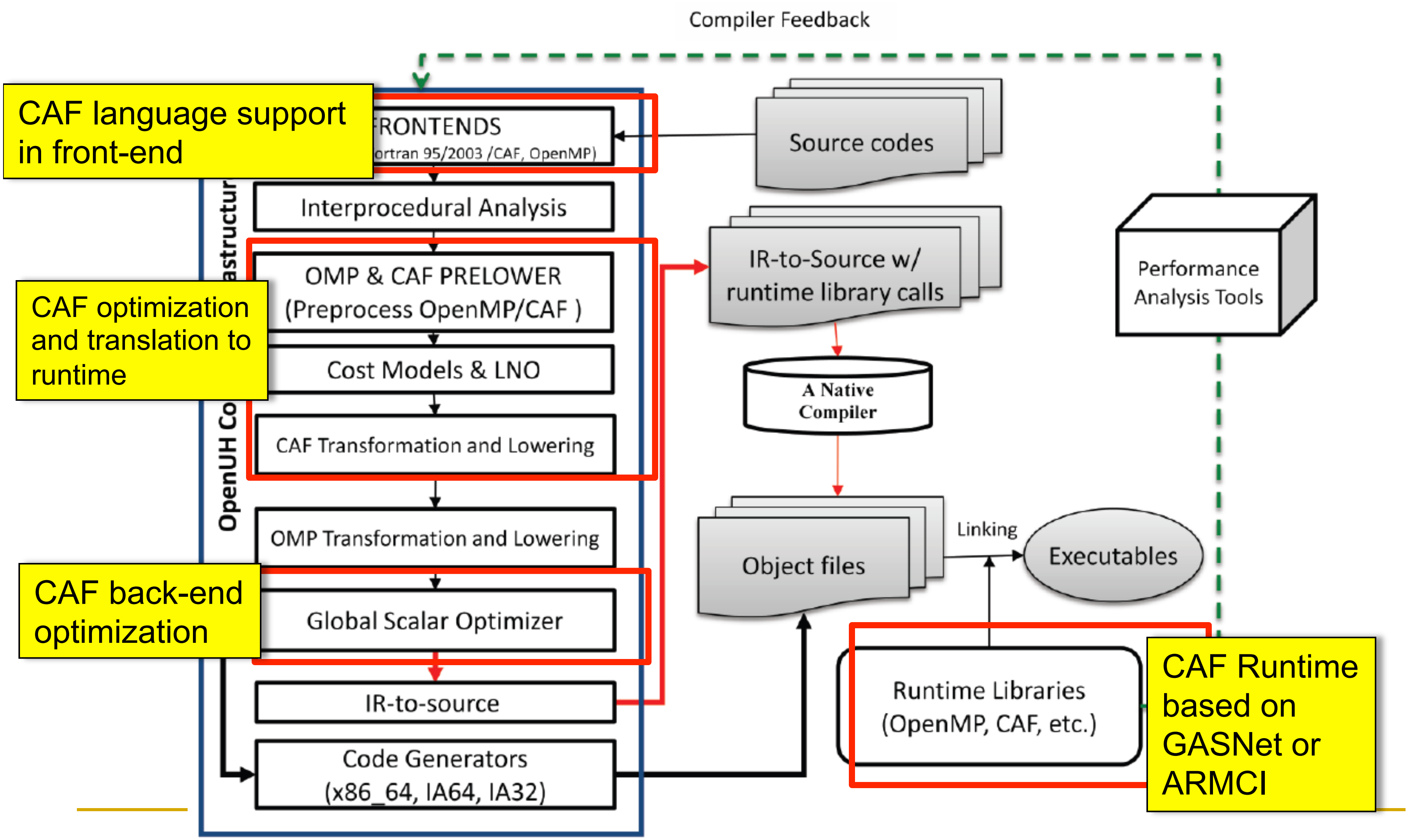




Operation	Energy (pJ)	DP FLOPs	Insts*
I$ Fetch	33	0.67	2.0
Register Access (3W)	10.5	0.2	0.6
Access 3 D$	100	2	6
Access 3 L2 D$	460	9	27
Access 3 off chip	762	15	45
Access 3 from DRAM	6000	120	360



CAF Support in OpenUH



Compiler Feedback

CAF language support in front-end

FRONTENDS

ortran 95/2003 /CAF, OpenMP)

Interprocedural Analysis

CAF optimization and translation to runtime

OMP & CAF PRELOWER (Preprocess OpenMP/CAF)

Cost Models & LNO

CAF Transformation and Lowering

OMP Transformation and Lowering

CAF back-end optimization

Global Scalar Optimizer

IR-to-source

Code Generators (x86_64, IA64, IA32)

OpenUH Co

astructur

Source codes

IR-to-Source w/ runtime library calls

A Native Compiler

Object files

Linking

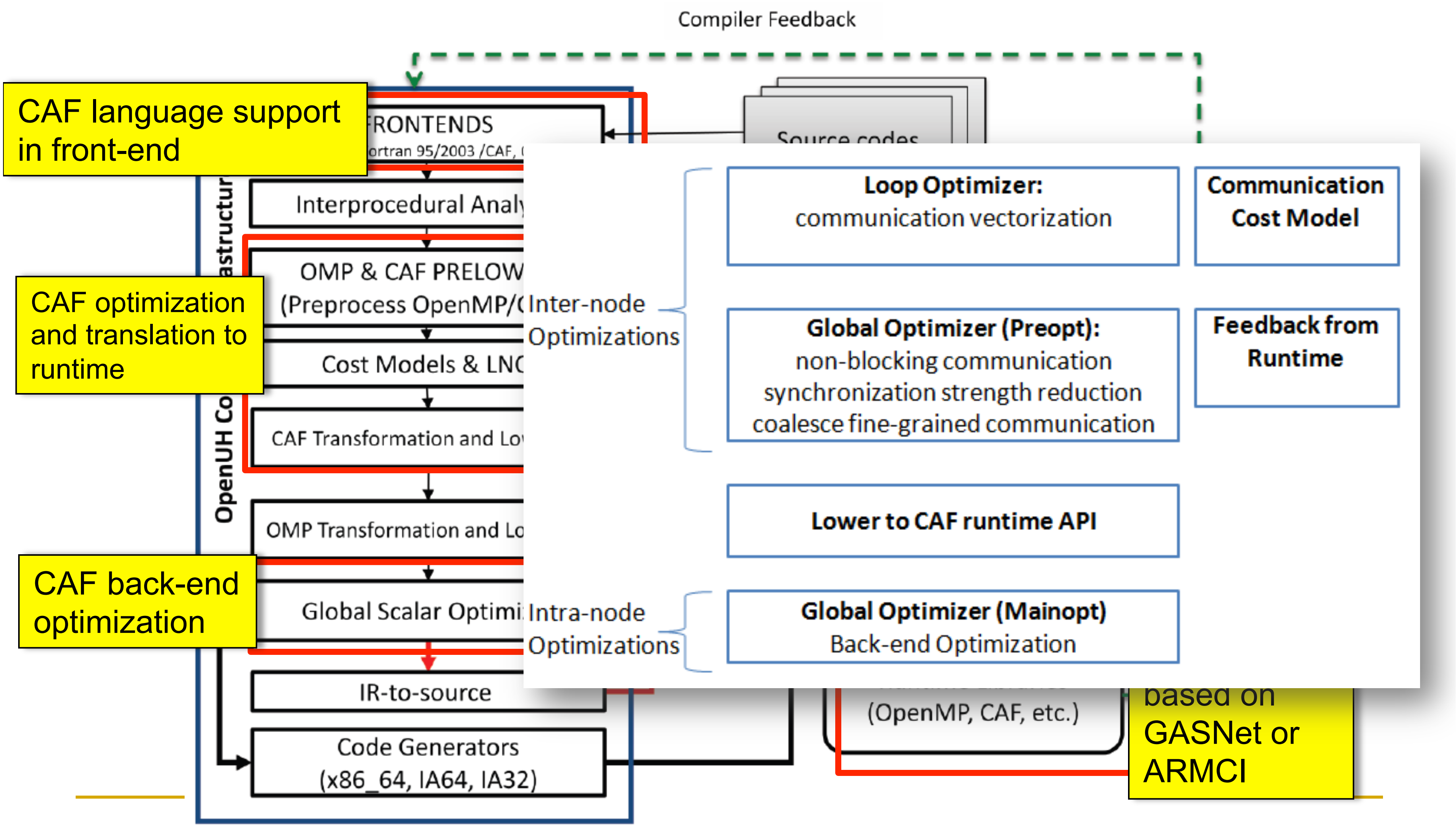
Executables

Performance Analysis Tools

Runtime Libraries (OpenMP, CAF, etc.)

CAF Runtime based on GASNet or ARMCI

CAF Support in OpenUH



Compiler Feedback

CAF language support in front-end

FRONTENDS

ortran 95/2003 /CAF, (

Source codes

Interprocedural Analy

OMP & CAF PRELOW

(Preprocess OpenMP/(

CAF optimization and translation to runtime

Cost Models & LNC

CAF Transformation and Lo

OMP Transformation and Lo

CAF back-end optimization

Global Scalar Optimi

IR-to-source

Code Generators

(x86_64, IA64, IA32)

OpenUH Co

astructur

(OpenMP, CAF, etc.)

based on GASNet or ARMCI

Inter-node Optimizations

Loop Optimizer: communication vectorization

Communication Cost Model

Global Optimizer (Preopt): non-blocking communication synchronization strength reduction coalesce fine-grained communication

Feedback from Runtime

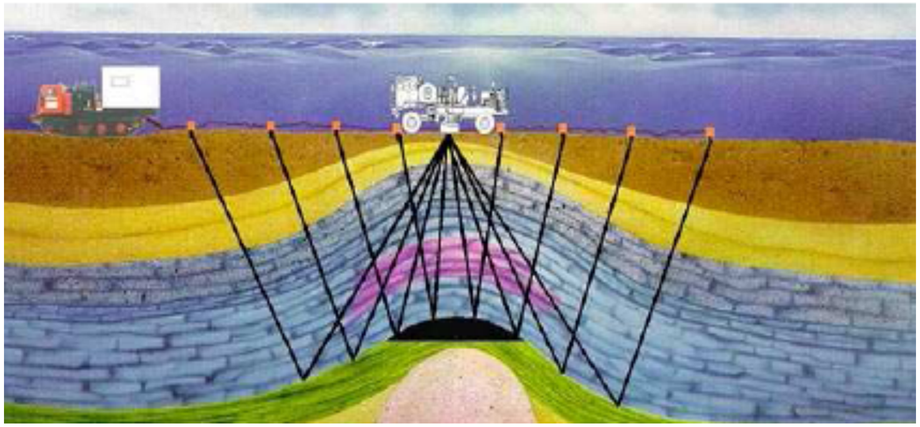
Lower to CAF runtime API

Intra-node Optimizations

Global Optimizer (Mainopt) Back-end Optimization

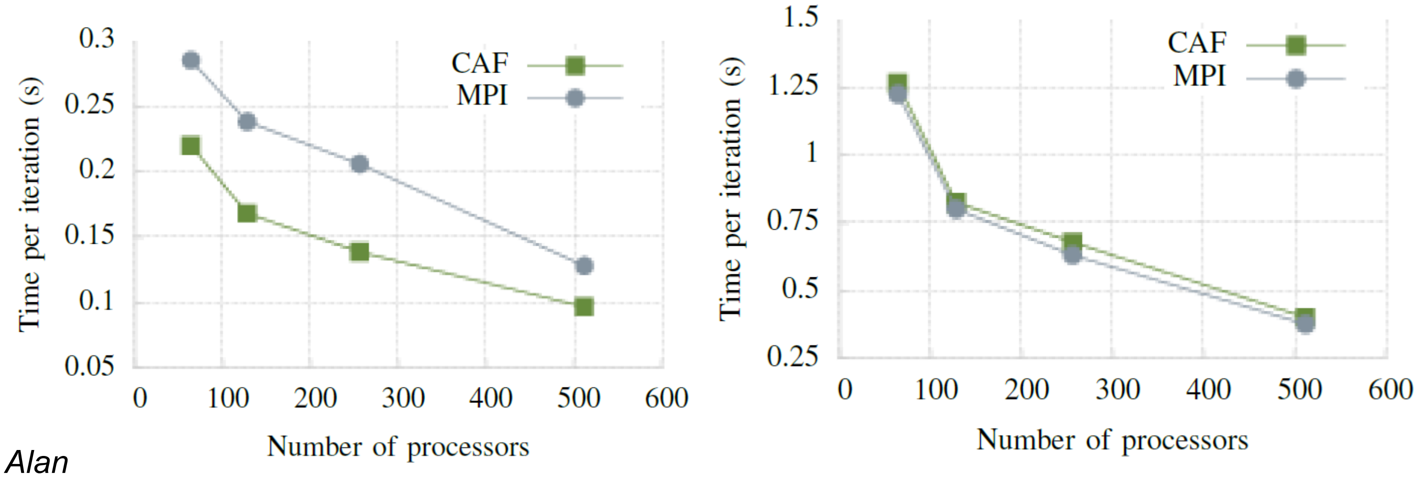
Reverse-time Migration Code in CAF

- A source wave is emitted per shot
- Reflected waves captured by array of sensors
- RTM (in time domain) uses finite difference method to numerically solve wave equation and reconstruct subsurface image (in parallel, with domain decomposition)



Forward Shot Comparison

Total Domain Size: 1024 x 768 x 512 (3.0 GB, per shot)
Comparison: OpenUH CAF, Intel MPI



(a) Isotropic

(b) TTI

CAF port and results by Alan Richardson, Summer 2012 Internship, Total.

PGAS Compiler Performance Optimization

- Reduce Messaging Overhead of Remote Accesses
  - aggregate fine-grained accesses; optimal message size is system dependent
  - identify communication patterns and convert 1-sided communication to optimized collective operations
- Reduce Round-trip Cost of Remote Accesses
  - generate split-phase access and use code motion to increase overlap of computation with in-progress remote accesses
  - transmit “codelets” to initiate a computation at the target rather than bring in data
- Reduce Synchronization Overhead
  - compiler analysis to identify over-synchronization
  - transformations to use split-phase barriers or point-to-point synchronization

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.

Alternative OpenMP Translation for Asynchronous Execution

- 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

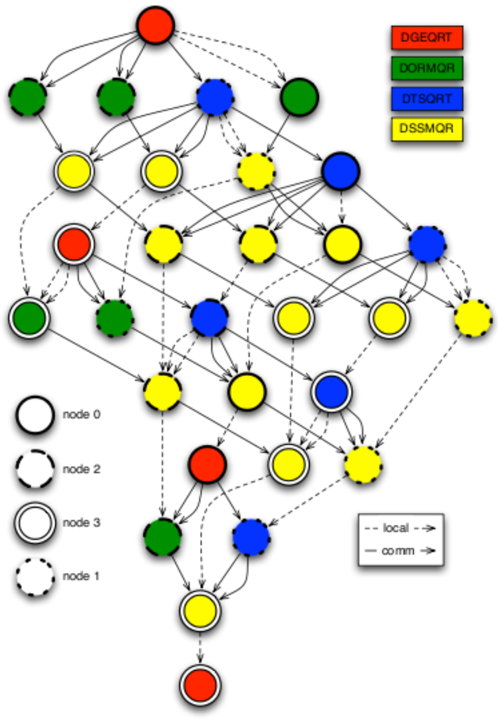
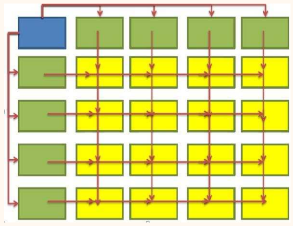
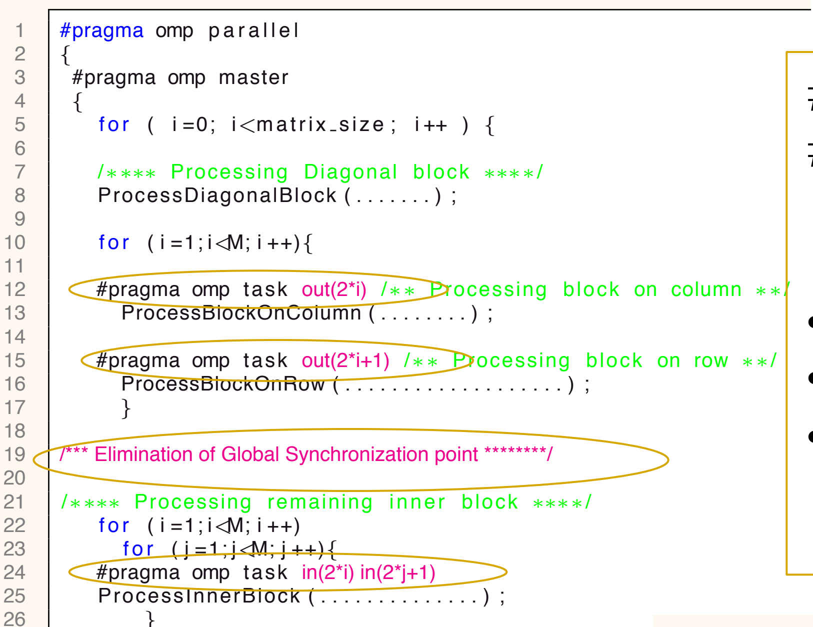


Fig. 5. DAG of QR for a 4x4 tile matrix.

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

IN|OUT|INOUT for Dependent Tasks



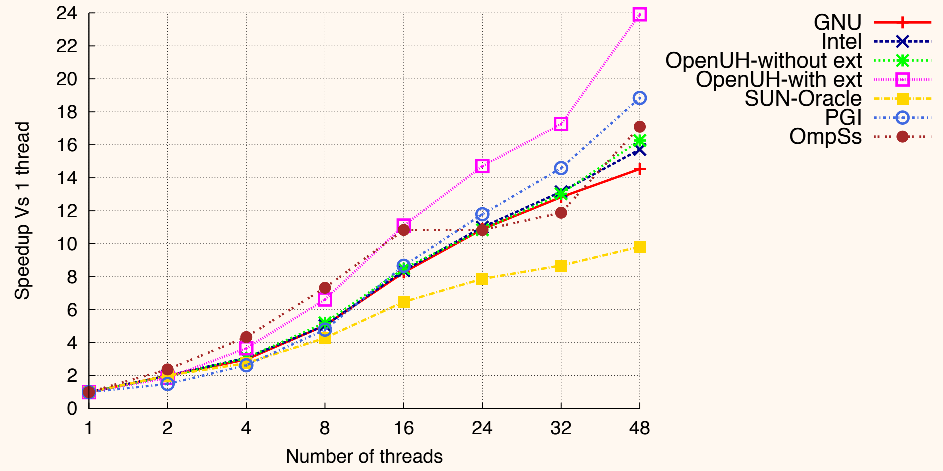


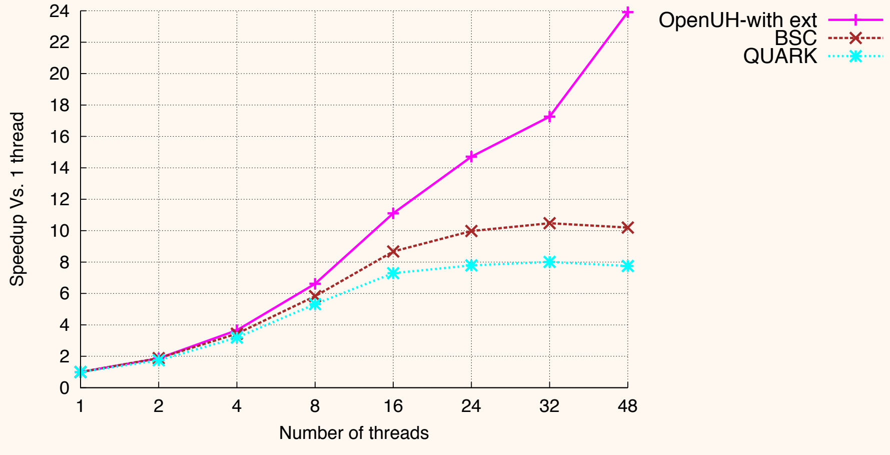
```
#pragma omp parallel
{
 #pragma omp master
 {
   for ( i=0; i<matrix_size; i++ ) {

   /**** Processing Diagonal block ****/
   ProcessDiagonalBlock(.......);

   for (i=1;i<M;i++){

  #pragma omp task out(2*i) /** Processing block on column **/
     ProcessBlockOnColumn(........);

  #pragma omp task out(2*i+1) /** Processing block on row **/
     ProcessBlockOnRow(...................);
     }

/*** Elimination of Global Synchronization point ********/

/**** Processing remaining inner block ****/
   for (i=1;i<M;i++)
     for (j=1;j<M;j++){
  #pragma omp task in(2*i) in(2*j+1)
   ProcessInnerBlock(..............);
      }
  #pragma omp taskwait
     }
```

#pragma omp task out [t1, t2, …]
#pragma omp task in [t1, t2, …]

Runtime

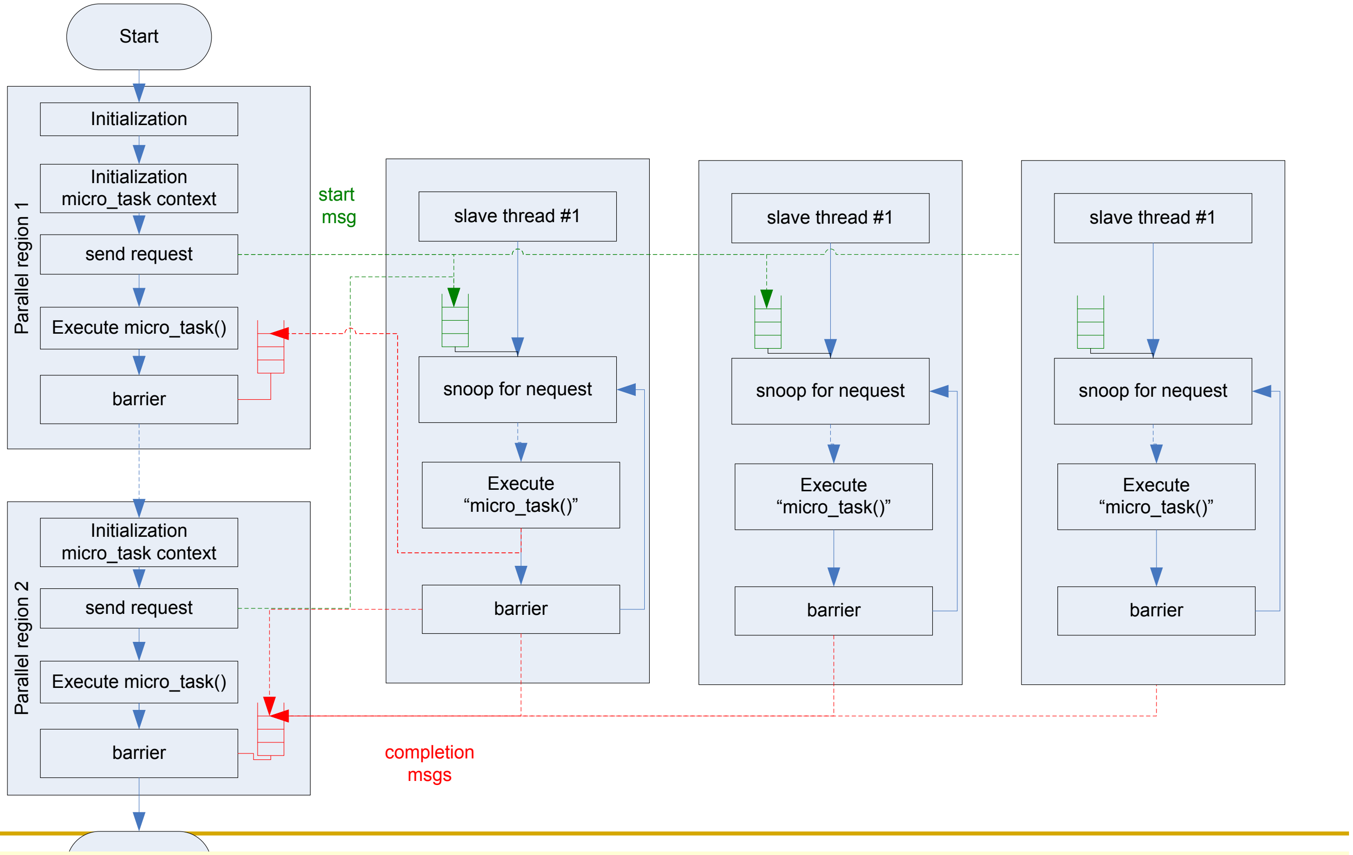
- Avoid the use of global locks
- Allows workstealing
- Decentralized dependence setup and resolution





Adapting Translation to New Kinds of Memory

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



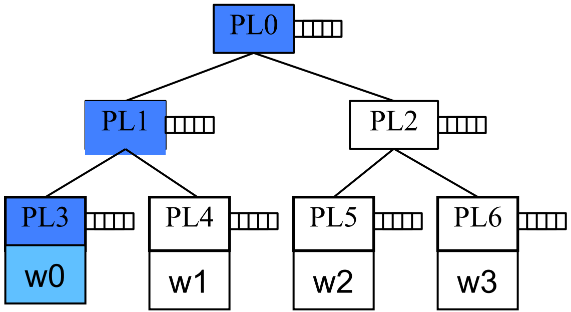
B. Chapman, L. Huang, E. Stotzer, E. Biscondi, A. Shrivastava, A. Gatherer. Implementing OpenMP on a High Performance Embedded Multicore MPSoC, pp 1-8, Proc. of Workshop on Multithreaded Architectures and Applications (MTAAP'09) In conjunction with International Parallel and Distributed Processing Symposium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

Agenda

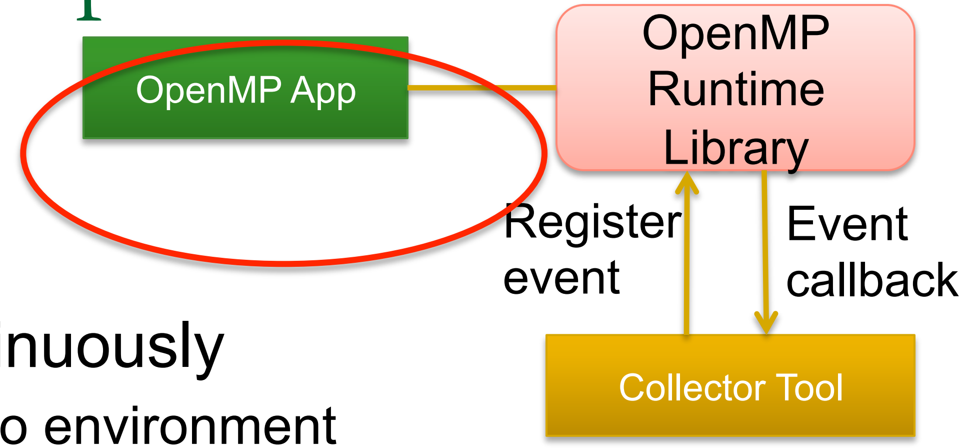
- The Challenge
- Programming Models: What’s Out There?
- Compiler Efforts: Increasing the Benefits
- **Runtime Support and System-Wide Interactions**

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

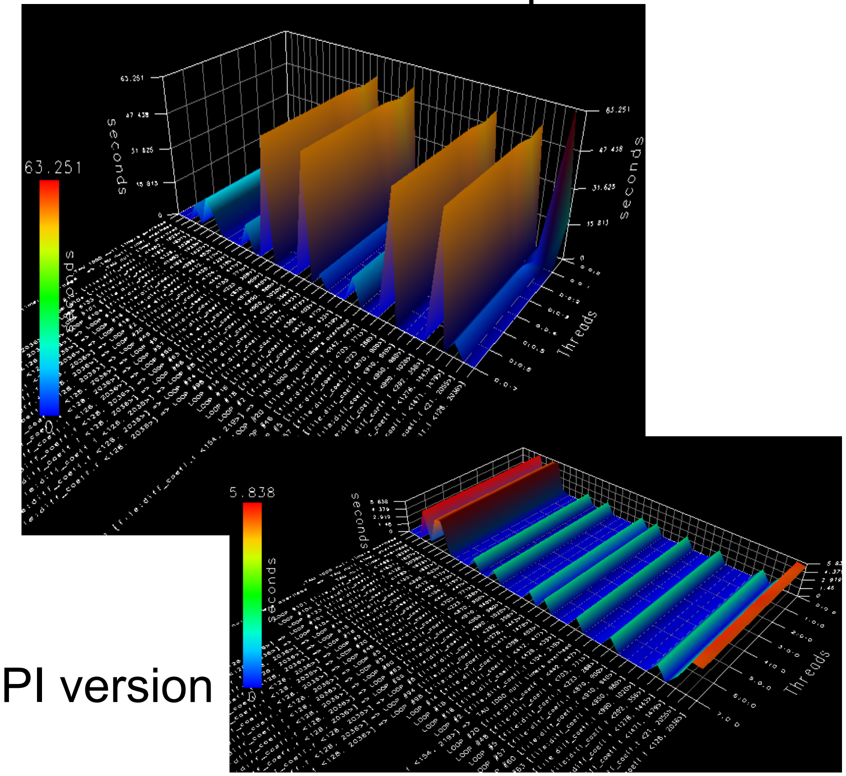


- Runtime support to continuously
  - Adapt workload and data to environment
  - Respond to changes caused by application characteristics, power, (impending) faults, system noise
  - Provide feedback on application behavior
- Collector Interface, implemented in compiler's runtime, enables monitoring 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

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

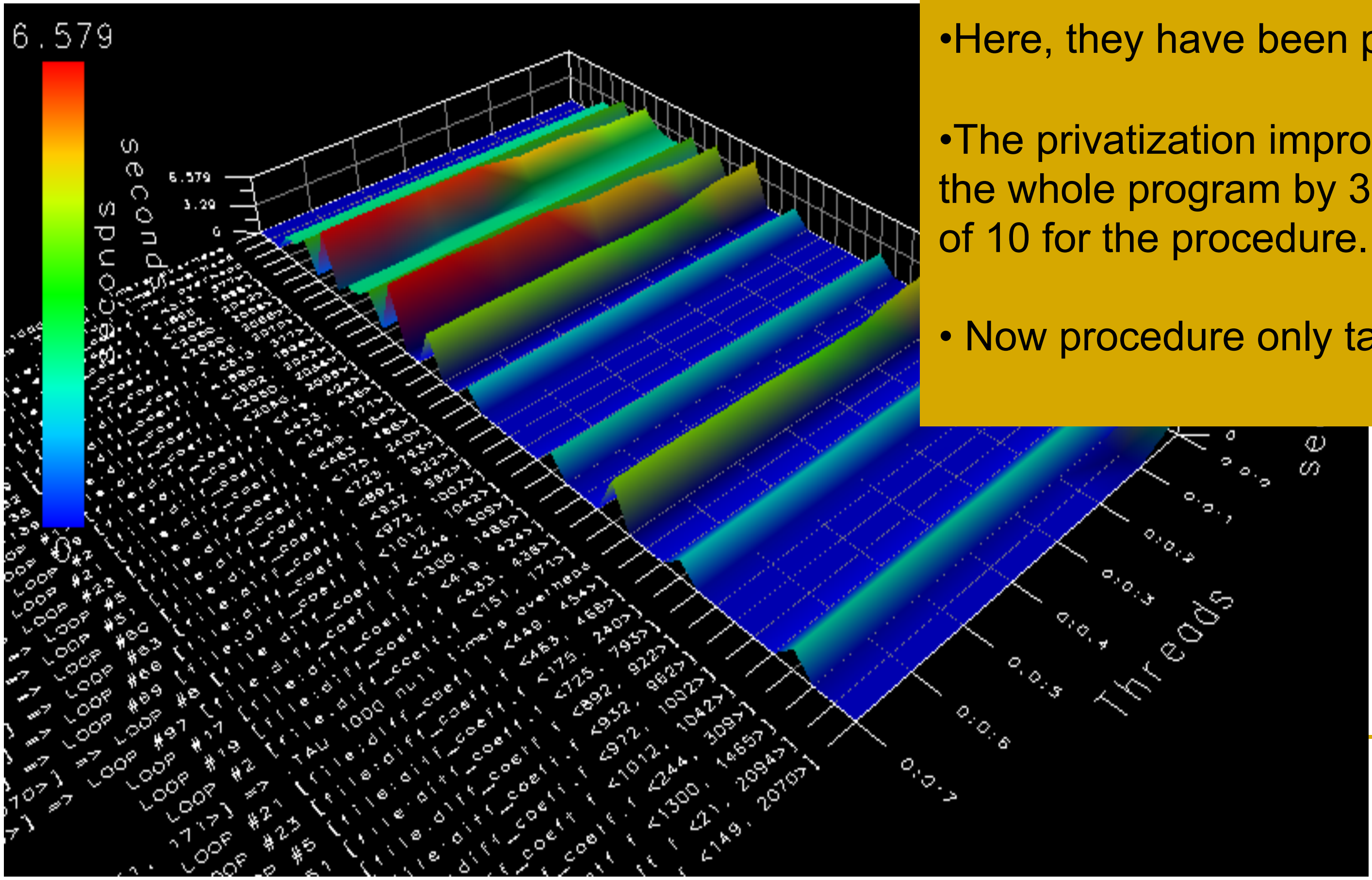


MPI version

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

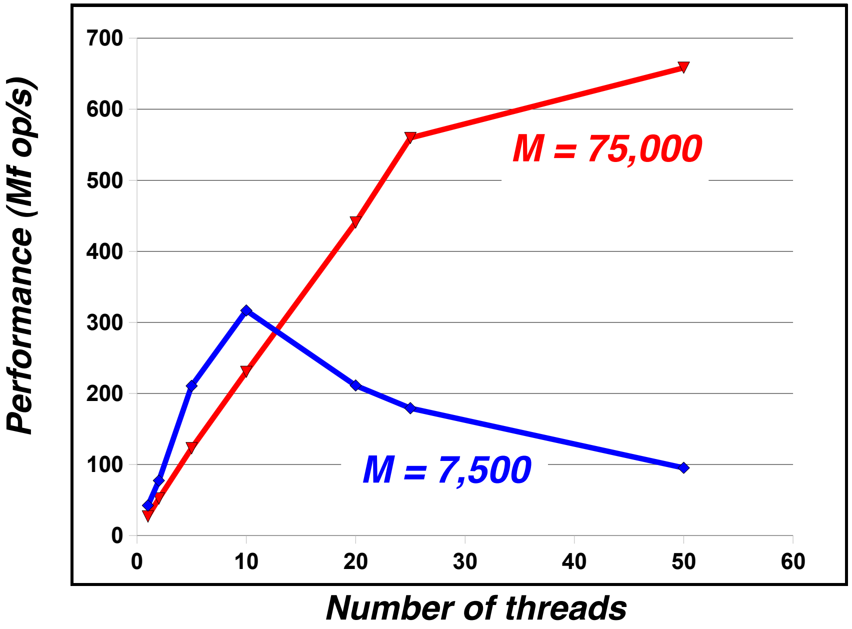
•Here, they have been privatized.

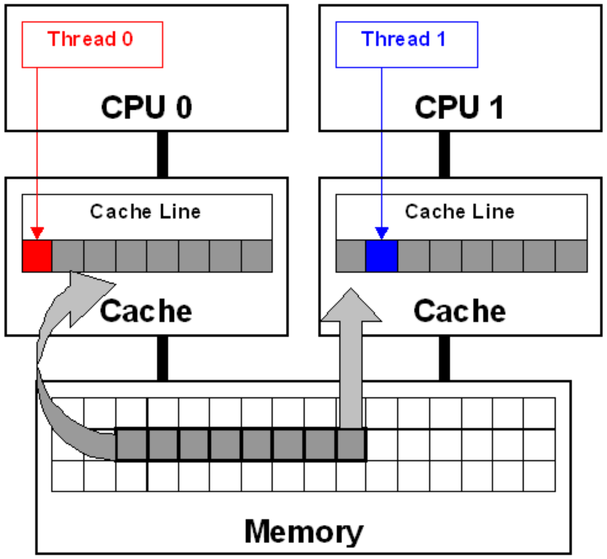
•The privatization improved the performance of the whole program by 30% and led to a speedup of 10 for the procedure.

• Now procedure only takes 5% of total time

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
```





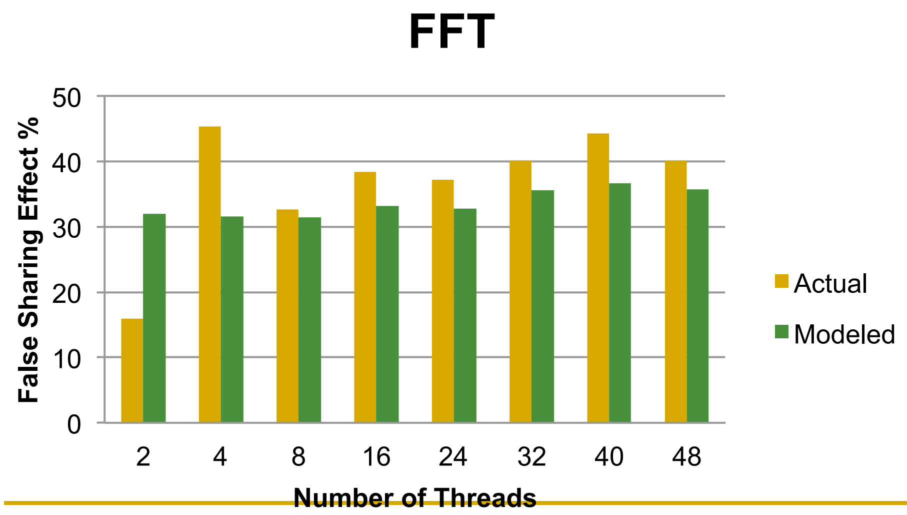
For a higher value of M, the program scales better

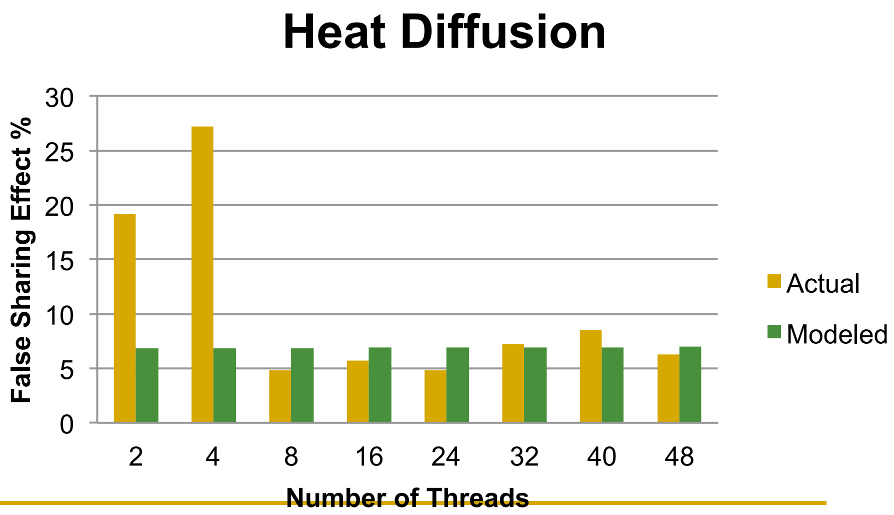
Modeling False Sharing at Compile-Time

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

False Sharing: Monitoring Results

- Cache line invalidation measurements (in Phoenix suite)

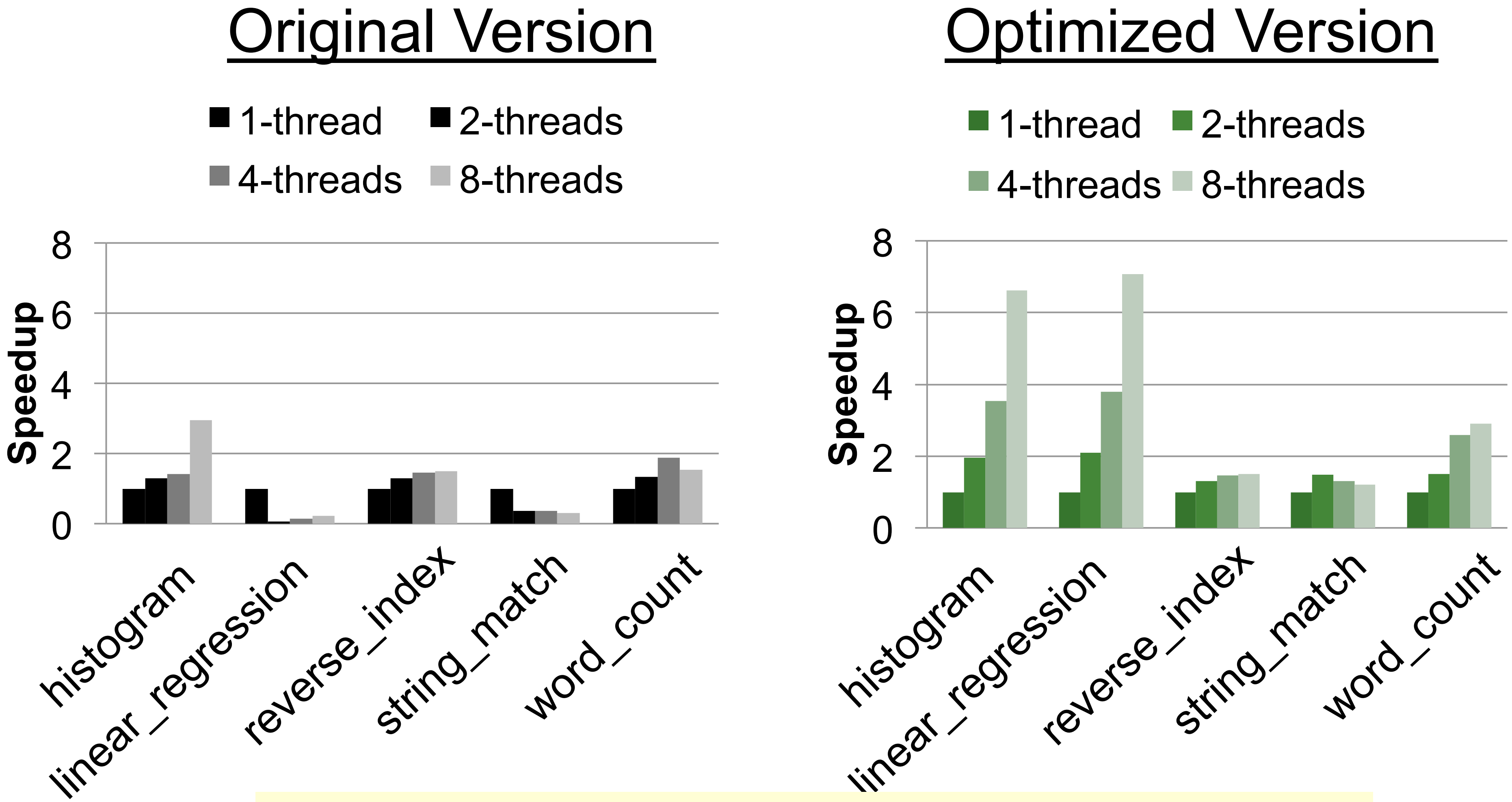
Program name	1-thread	2-threads	4-threads	8-threads
histogram	13	**7,820,000**	**16,532,800**	**5,959,190**
kmeans	383	28,590	47,541	54,345
linear_regression	9	**417,225,000**	**254,442,000**	**154,970,000**
matrix_multiply	31,139	31,152	84,227	101,094
pca	44,517	46,757	80,373	122,288
reverse_index	4,284	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	-

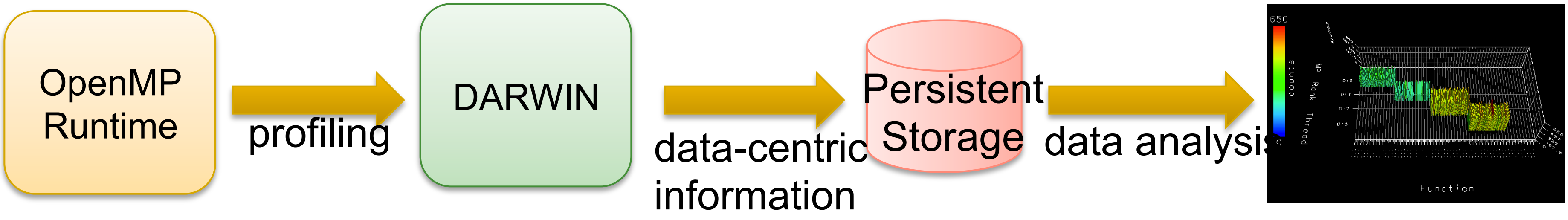
Runtime False Sharing Detection



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

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



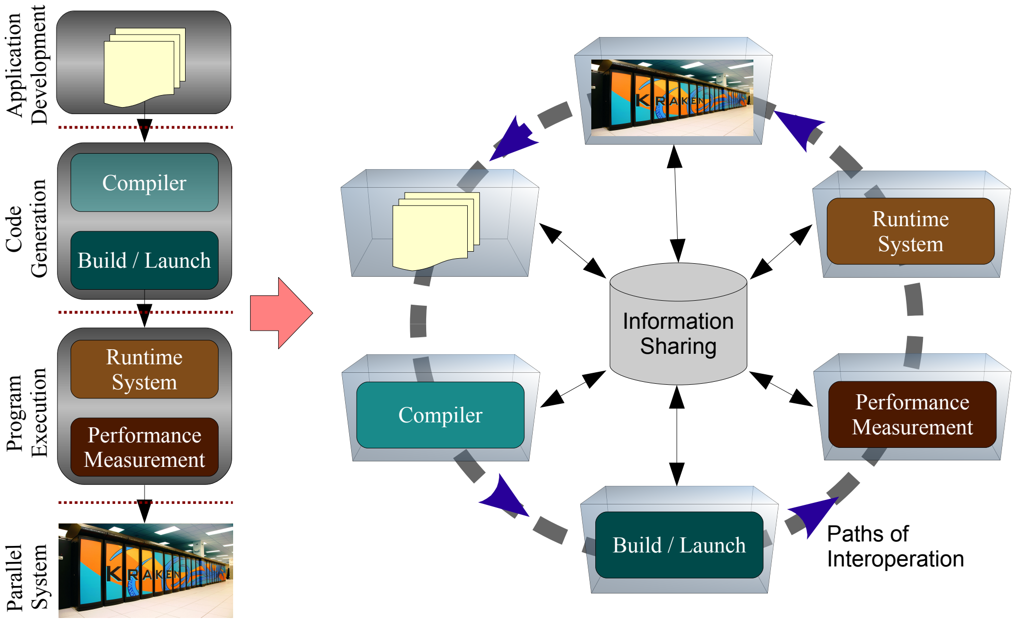
OpenMP Runtime → profiling → DARWIN → data-centric information → Persistent Storage → data analysis

Besar Wicaksono, Ramachandra C Nanjegowda, and Barbara Chapman. A Dynamic Optimization Framework for OpenMP. IWOMP 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
- Use with architectural information, system state, smart monitoring for adaptation on the fly
- Compiler modeling for dynamic optimization as well as feedback to user, tools
- And much more…

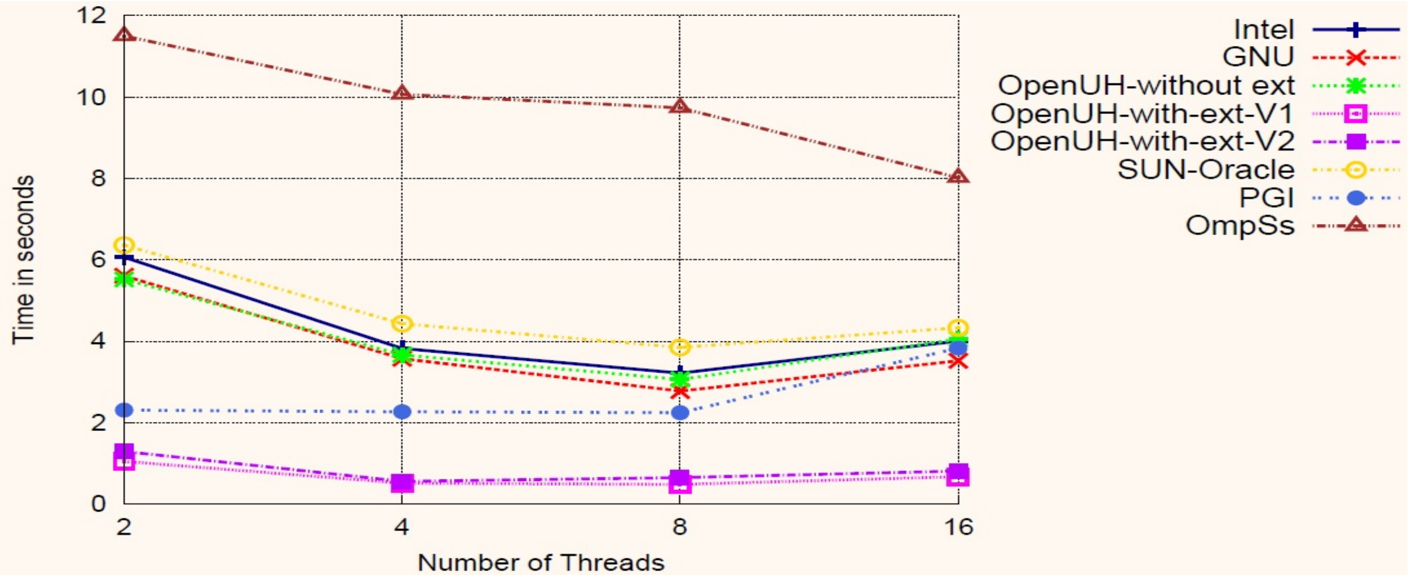


Additional Slides

- BACKUP SLIDES FROM HERE

Results – Smith Waterman (-O0 optimization) Sequence Size: 4096

Performance (in secs) with task chunk size 320, w.r.t. commercial compilers :



Performance (in secs) with task chunk size 320 w.r.t. related dataflow models:

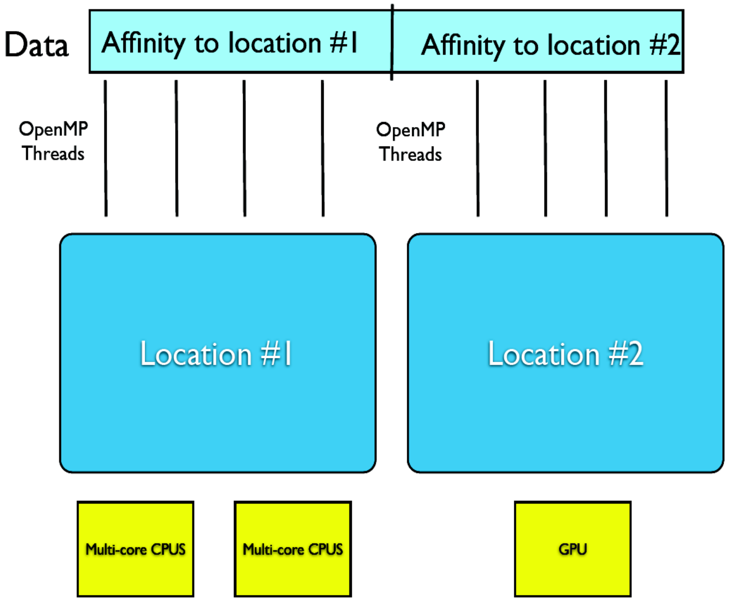
Threads	OpenUH_ext	OmpSs_ext	Quark
2	1.045	52.251	2.639
4	0.511	50.640	2.278
8	0.480	48.645	2.081
16	0.669	46.256	2.395

Benchmark Suite Results using K20

Applications	Domains	OpenACC Directive Combinations	Lines of Code Added vs Serial		Speedup Over		
			OpenMP	OpenACC	Seq	OpenMP	CUDA
Needleman-Wunsch	Bioinformatics	data copy, copyin kernels present loop gang, vector, private	6	5	2.98	1.28	0.24
Stencil	Cellular Automation	data copyin, copy, deviceptr kernels present loop collapse, independent	1	3	40.55	15.87	0.92
Computational Fluid Dyanmics (CFD)	Fluid Mechanics	data copyin, copy, deviceptr data present, deviceptr kernels deviceptr kernels loop, gang, vector, private loop gang, vector acc_malloc(), acc_free()	8	46	35.86	4.59	0.38
2D Heat (grid size 4096*4096)	Heat Conduction	data copyin, copy, deviceptr kernels present loop collapse, independent	1	3	99.52	28.63	0.90
Clever (10Ovals)	Data Mining	data copyin kernels present, create, copyin, copy loop independent	10	3	4.25	1.22	0.60
FeldKemp (FDK)	Image Processing	kernels copyin, copyout loop, collapse, independent	1	2	48.30	6.51	0.75

OpenMP Locality Research

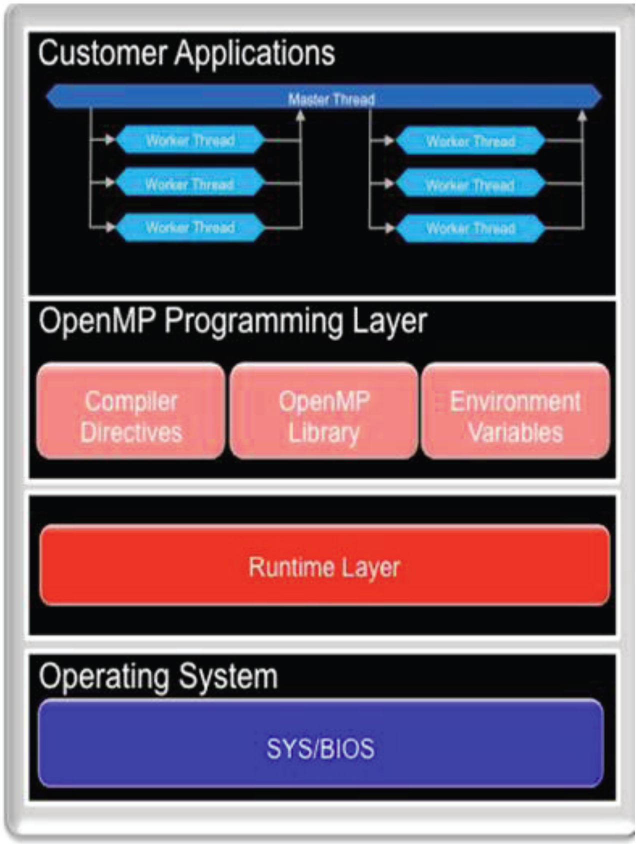
Locations := Affinity Regions, Based on Locales, Places



- 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

TI's OpenMP Solution Stack



- Leverages the DSP shared memory architecture for multicore.
- Makes use of dedicated DSP hardware for fast synchronization.
- Runs efficiently on TI's SYS/BIOS™ realtime operating system (RTOS).

TI's OpenMP solution stack. The C66x compiler translates OpenMP into multi-threaded code with calls to the runtime API. The runtime library is implemented on top of distributed copies of SYS/BIOS and IPC.