Portability in an Age of Node Diversity

Can our programming models cope?

Barbara Chapman

University of Houston

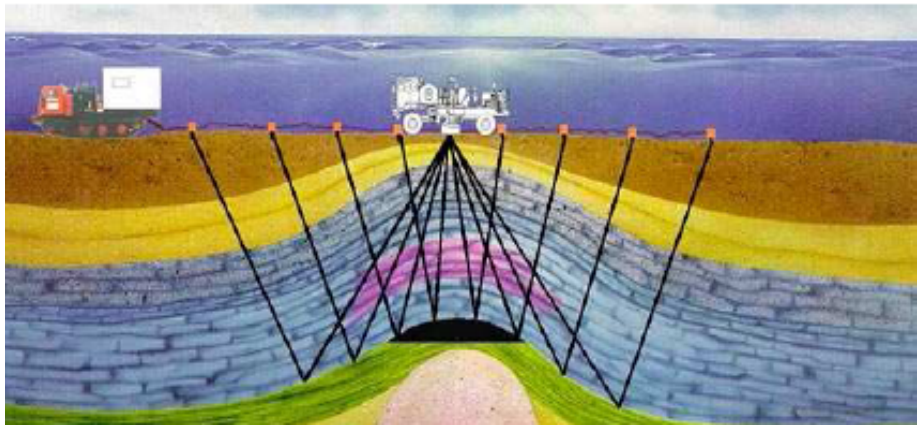
CCDSC Lyon; September 2014

Acknowledgements: NSF CNS-0833201, CCF-0917285; DOE DE-FC02-06ER25759, ORAU, Shell, Total, Texas Instruments, u.a.

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

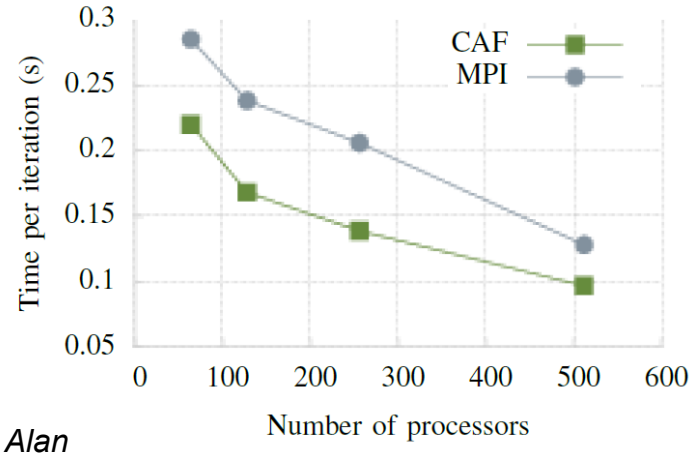
Porting Reverse-time Migration Code to CAF

- 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)

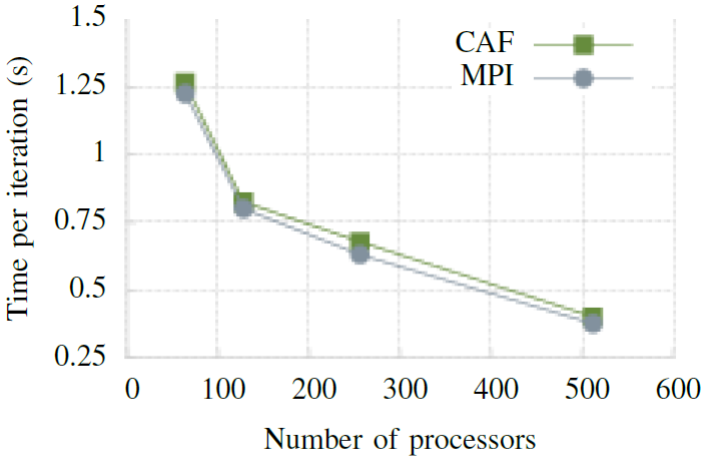


Forward Shot Comparison

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



(a) Isotropic



(b) TTI

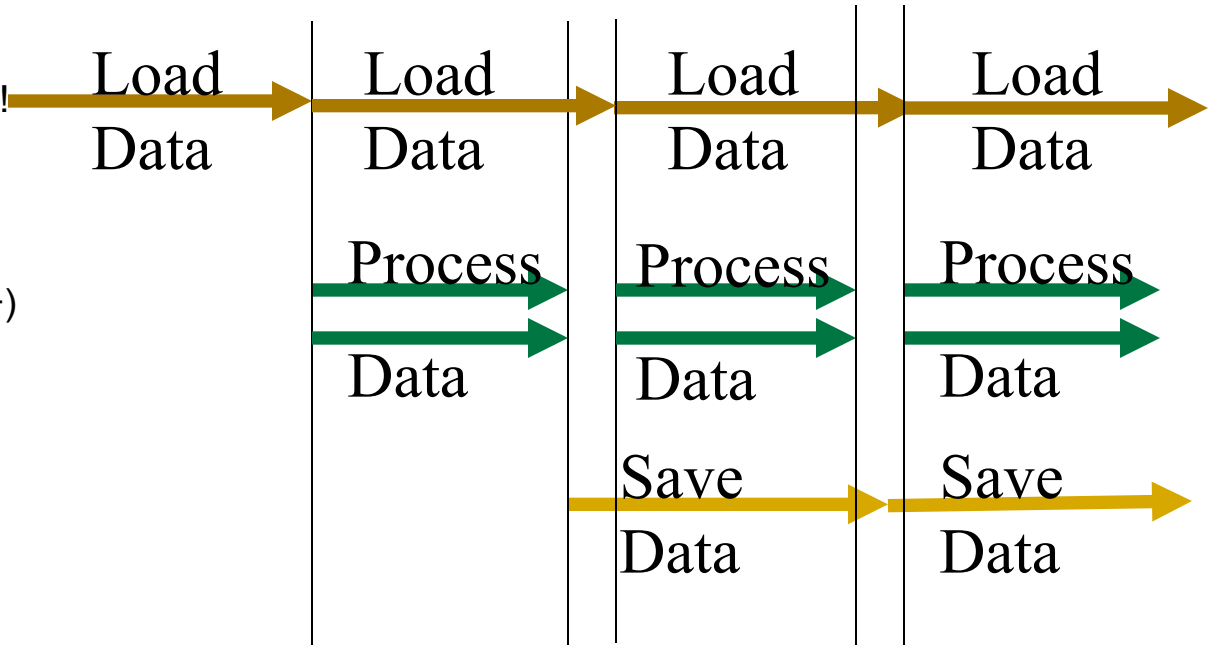
CAF port and results by Alan Richardson, Summer 2012 Internship, Total.


We can talk to the animals.
Because HP speaks Big Data.
We can talk to the animals.
Because HP speaks Big Data.


We can talk to the animals.
We can talk to the animals.
Because HP speaks Big Data.
We can talk to the animals.

Seismic Data Processing in OpenMP

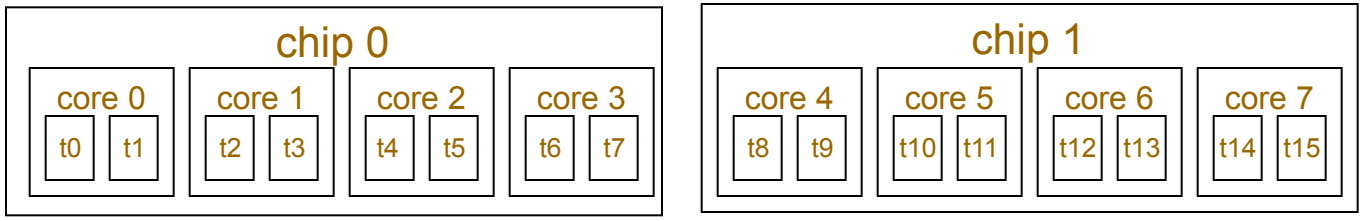
```
 Loadline(nStartLine,...);  // preload the first line of data
#pragma omp parallel
{
   for (int iLineIndex=nStartLine; iLineIndex <= nEndLine; iLineIndex++)
   {
#pragma omp single nowait
  {// loading the next line data, NO WAIT!
        Loadline(iLineIndex+1,...);
   }
#pragma omp for schedule(dynamic)
      for(j=0;j<iNumTraces;j++)
            for(k=0;k<iNumSamples;k++)
                  processing();
#pragma omp single nowait
     {
        SaveLine(iLineIndex);
     }
   }
}
```



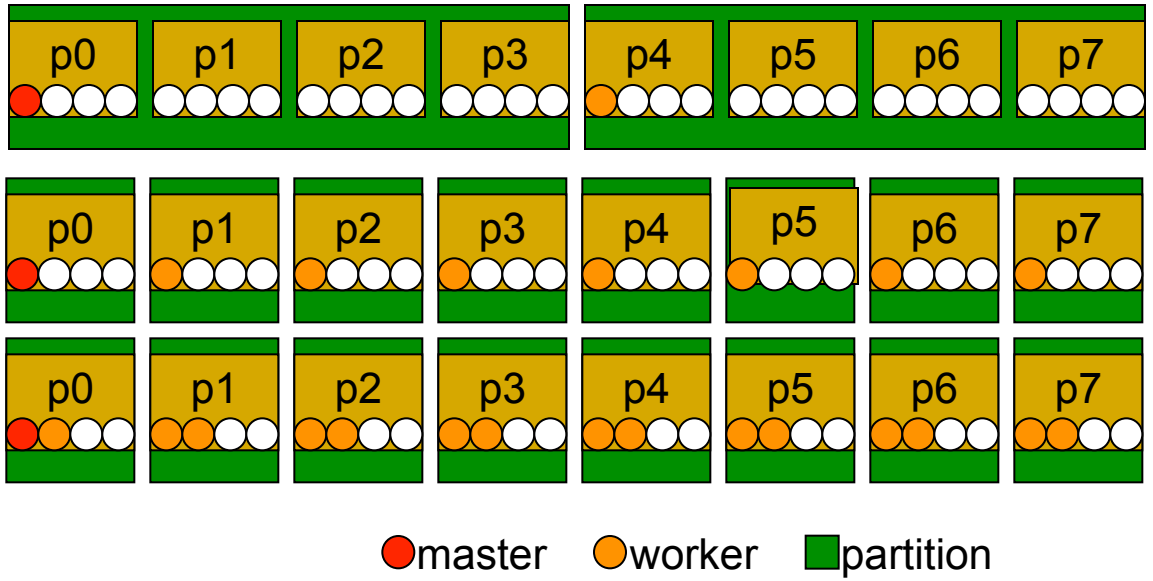
OpenMP 4.0

- Released July 2013
  - http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
  - http://www.openmp.org/mp-documents/OpenMP_Examples_4.0.1.pdf
- Main changes from 3.1:
  - Accelerator extensions
  - SIMD extensions
  - Places and thread affinity
  - Taskgroup and dependent tasks
  - Error handling (cancellation)
  - User-defined reductions

Thread Affinity

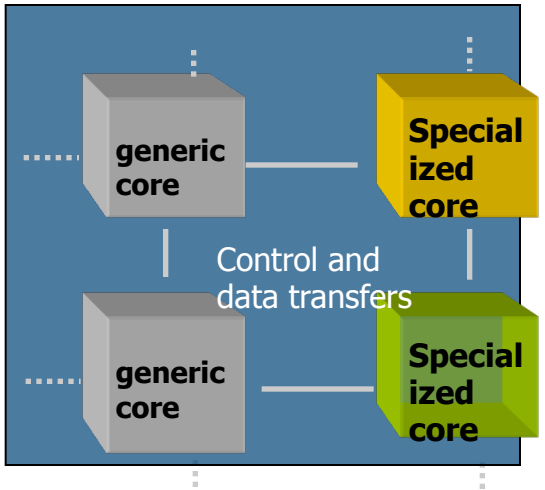


- Mapping and binding of OpenMP threads
- “proc_bind(spread)”
  - spread 2
  - spread 8
  - spread16



It's A Heterogeneous World

- High-level directive-based parallel programming
- OpenMP as a unified, productive programming model for heterogeneous nodes
  - Principles identifed by PGI, CAPS
- Identify code to run on certain kind of core
- Where and when is data allocated?
- How to optimize data motion?



HMPP

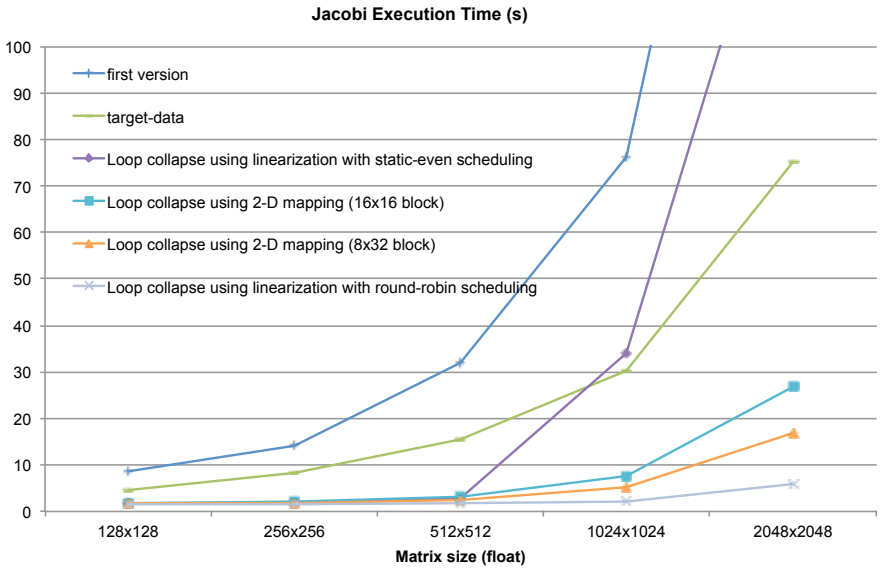
```
//acquire a device
#pragma hmpp fdtd acquire
//allocate data on the device
#pragma hmpp fdtd allocate
#pragma hmpp fdtd allocate, data["in"]"out"], data["in"]"out"].size={dx*dy*dz}, &
#pragma hmpp &              data["in"]"out"].elementsize="sizeof(double)"
//upload of data based on the address - 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

OpenMP for Accelerators

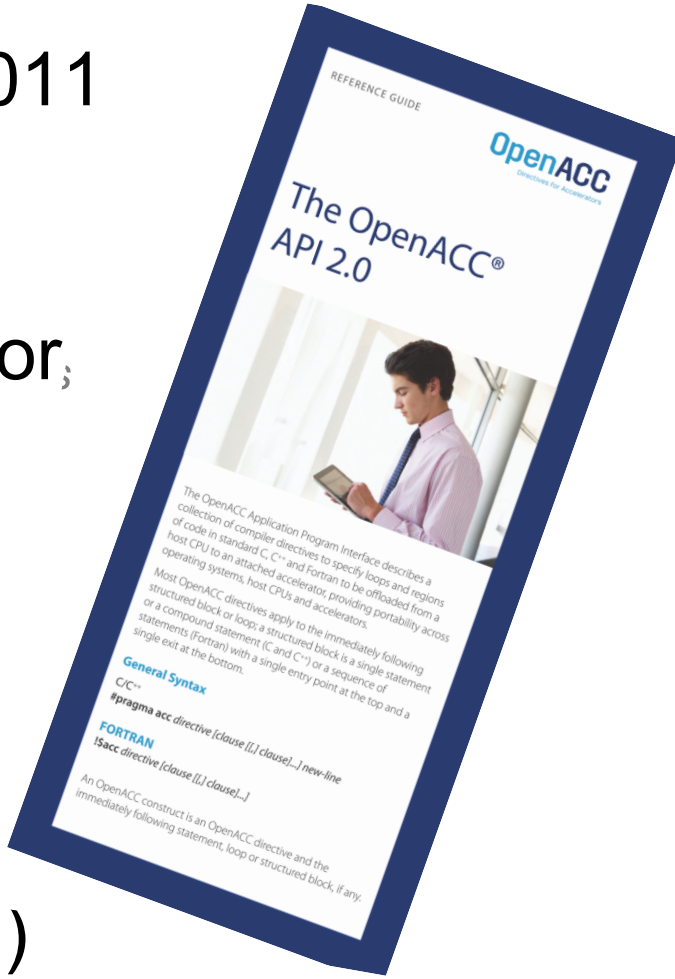
```
#pragma omp target data device (gpu0) map(to:n, m, omega, ax, ay, b, \
   f[0:n][0:m])  map(tofrom:u[0:n][0:m]) map(alloc:uold[0:n][0:m])
while ((k<=mits)&&(error>tol))
{
// a loop copying u[][] to uold[][] is omitted here
 …
#pragma omp target  device(gpu0)

#pragma omp parallel for private(resid,j,i) reduc
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
    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

OpenACC Programming Model

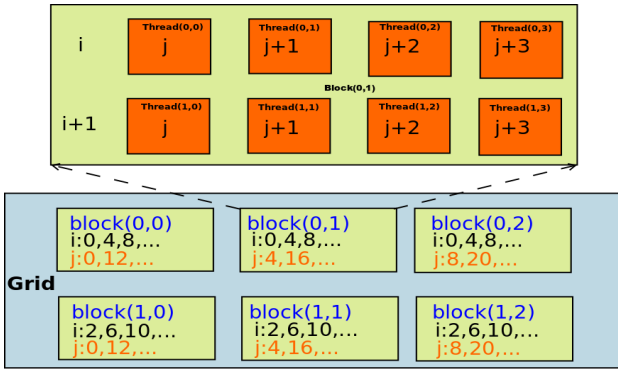
- Announced Supercomputing 2011
  - Initial work by NVIDIA, Cray, PGI, CAPS
- Directive-based programming for accelerators
  - For Fortran, C, C++
  - Loop-based computations
- Current version 2.0
- Compilers: PGI, Cray, CAPS, OpenUH, OpenARC, GCC (4.9)



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++ ) {..... }
}
```



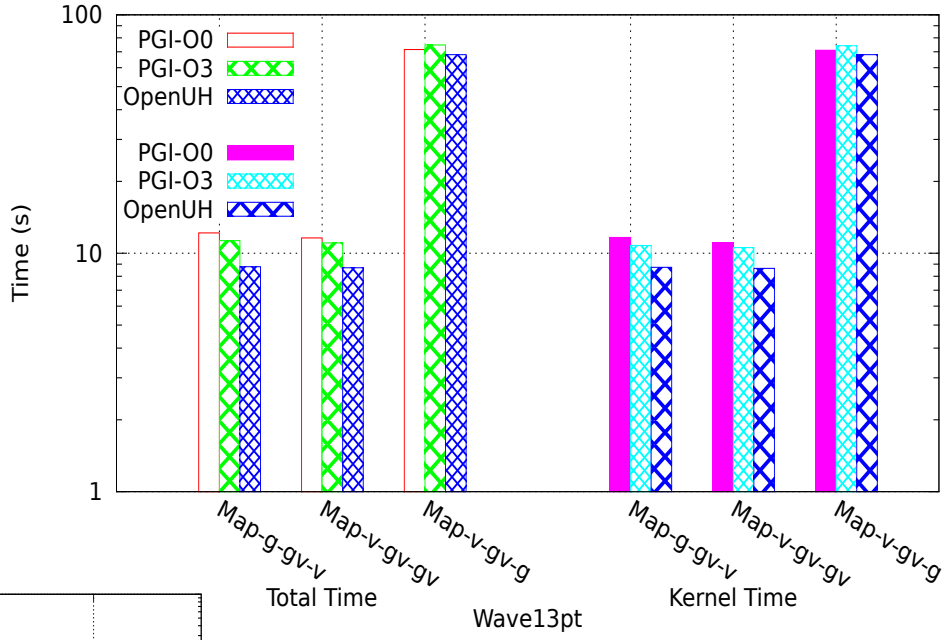


Figure: Wave13pt

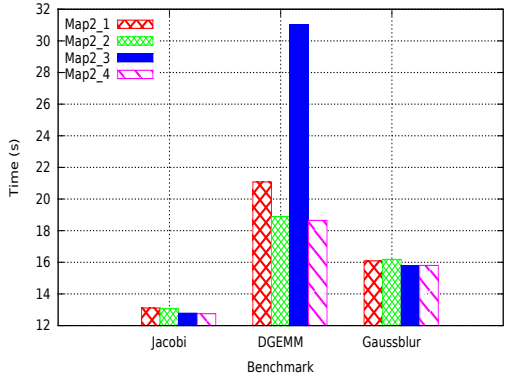


Fig. 9: Double nested loop mapping.

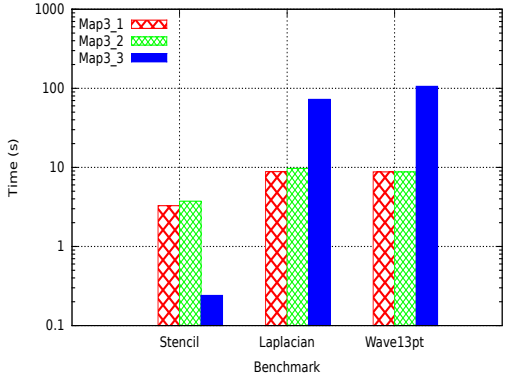


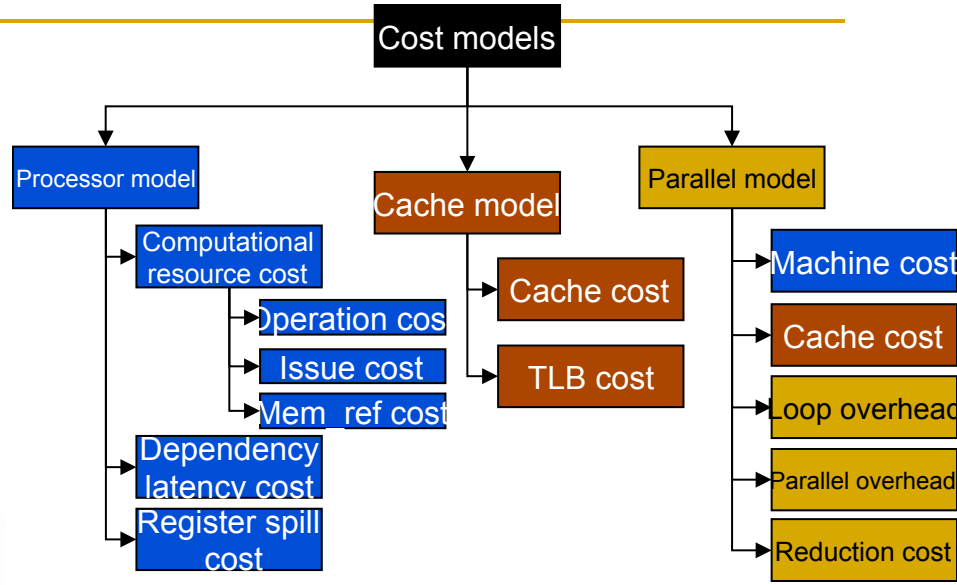
Fig. 10: Triple nested loop mapping.

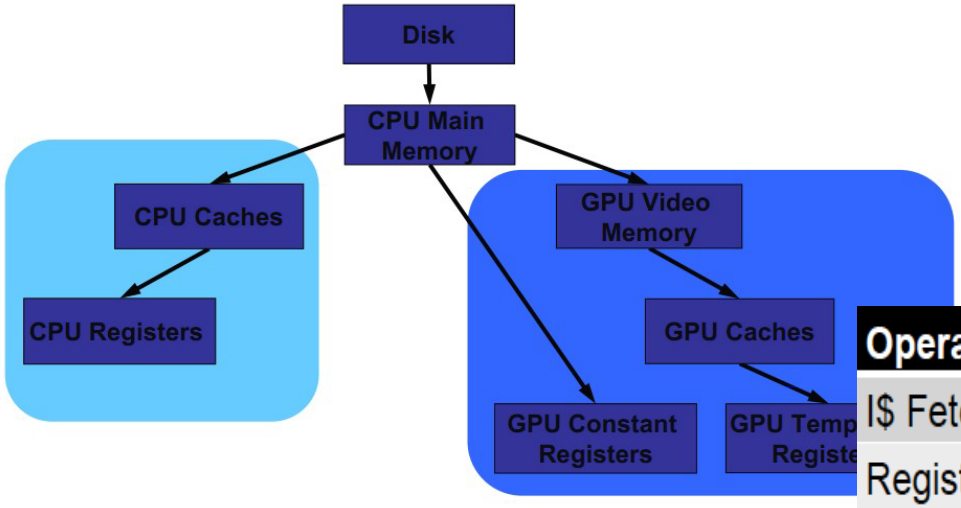
Compiling a High-level Directive-Based Programming Model for GPGPUs; Xiaonan Tian, Rengan Xu, Yonghong Yan, Zhifeng Yun, Sunita Chandrasekaran, and Barbara Chapman; 26th International Workshop on Languages and Compilers for Parallel Computing (LCPC2013)

OpenACC Status

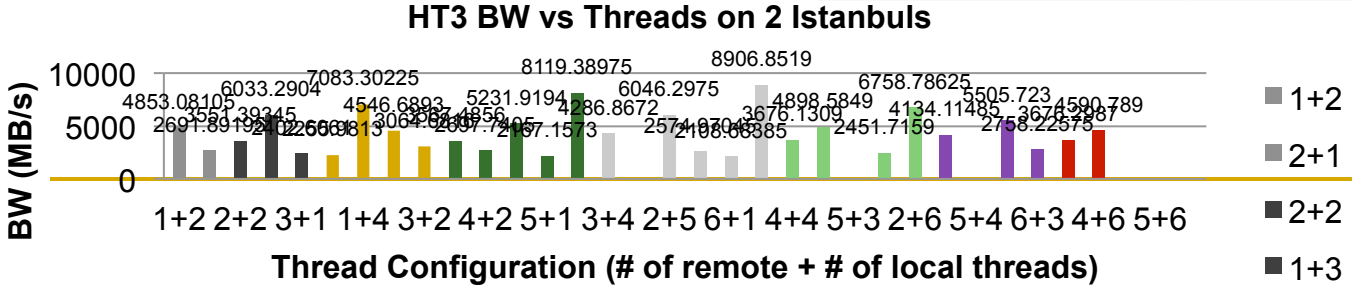
- Under active development
- Significant extensions in version 2.0 include:
  - Procedure calls, separate compilation
  - Nested parallelism
  - Loop tile clause
  - Device resident global data
  - New atomic construct
- Future plans include:
  - Tools interface
  - Deep Copy for pointer based data structures
  - Better performance portability across implementations


Compiler Models



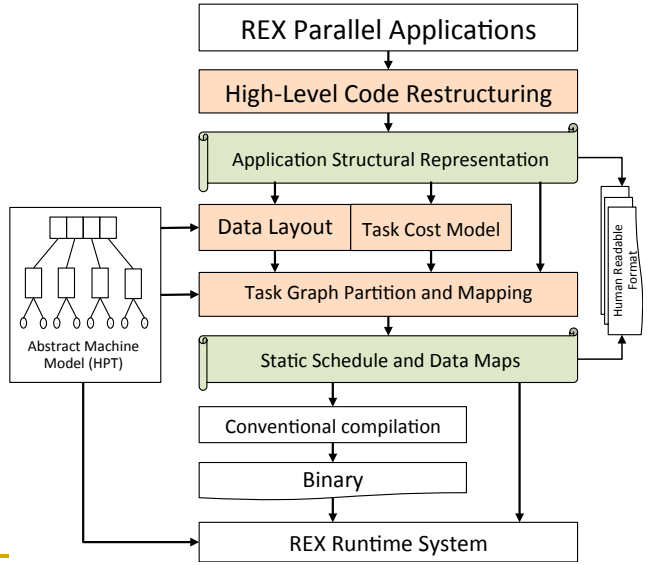


Operation	Energy (pJ)	DP FLOPs	Insts*
I$ Fetch	33	0.67	2.0
Register Access (3W)	10.5	0.2	0.6
Access 3 D$	100	2	6
Access 3 L2 D$	460	9	27
Access 3 off chip	762	15	45
Access 3 from DRAM	6000	120	360



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.

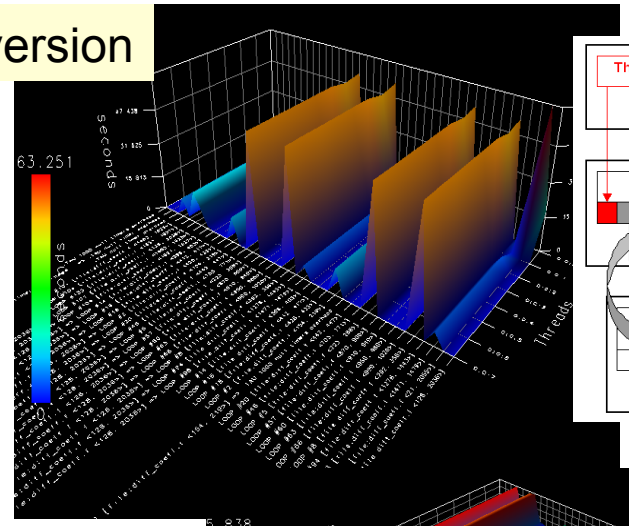


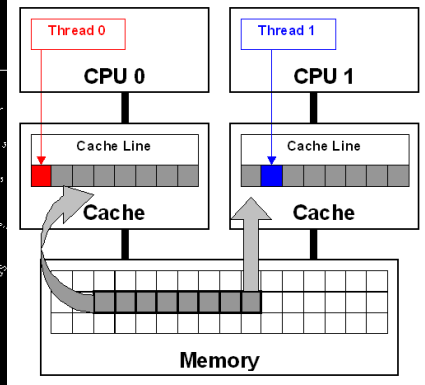
Is there a generic code structure that supports this process?

Small "Mistakes", Big Consequences

- GenIDLEST code
  - Solves incompressible Navier Stokes and energy equations

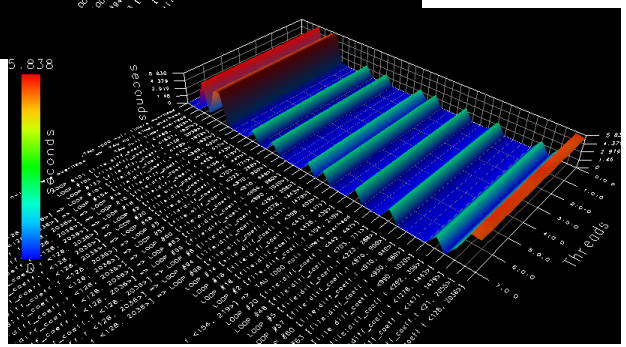
OpenMP version





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

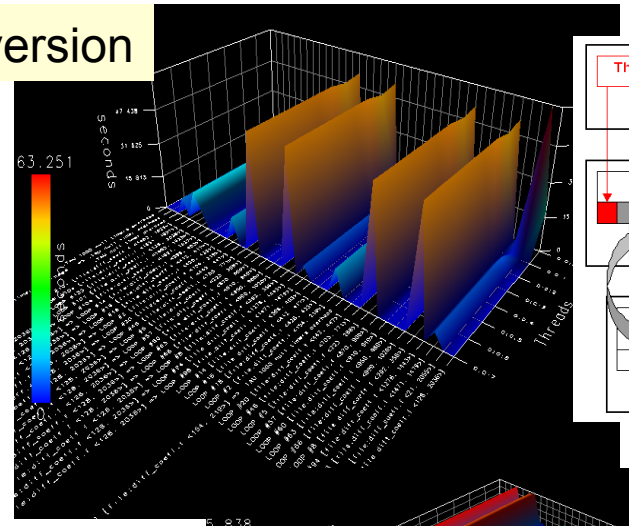
MPI version

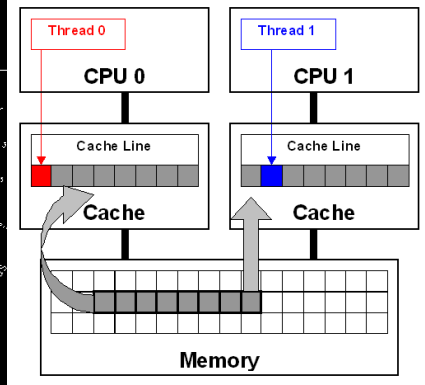


Small "Mistakes", Big Consequences

- GenIDLEST code
  - Solves incompressible Navier Stokes and energy equations

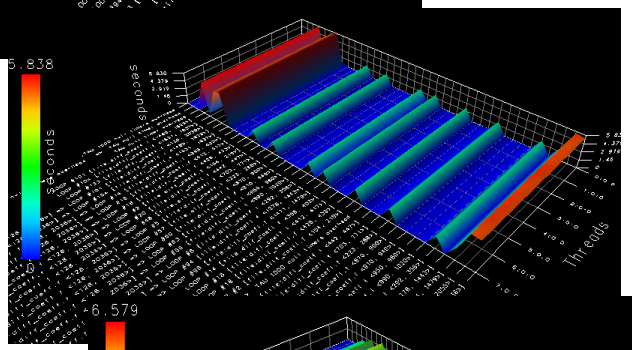
OpenMP version



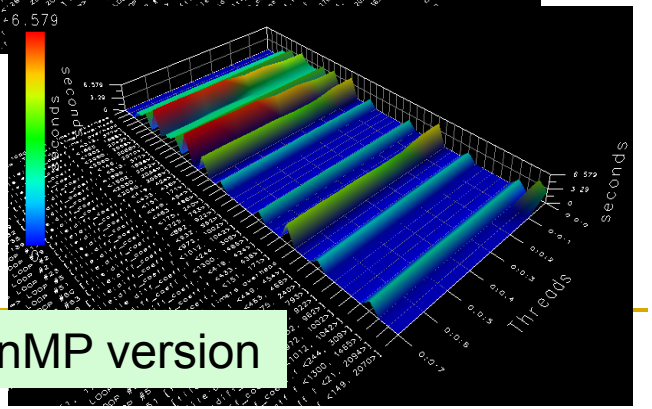


OpenMP version: a single procedure is responsible for 20% of total time, is 9 times slower than MPI version . Its loops are up to 27 times slower in OpenMP than MPI.

MPI version

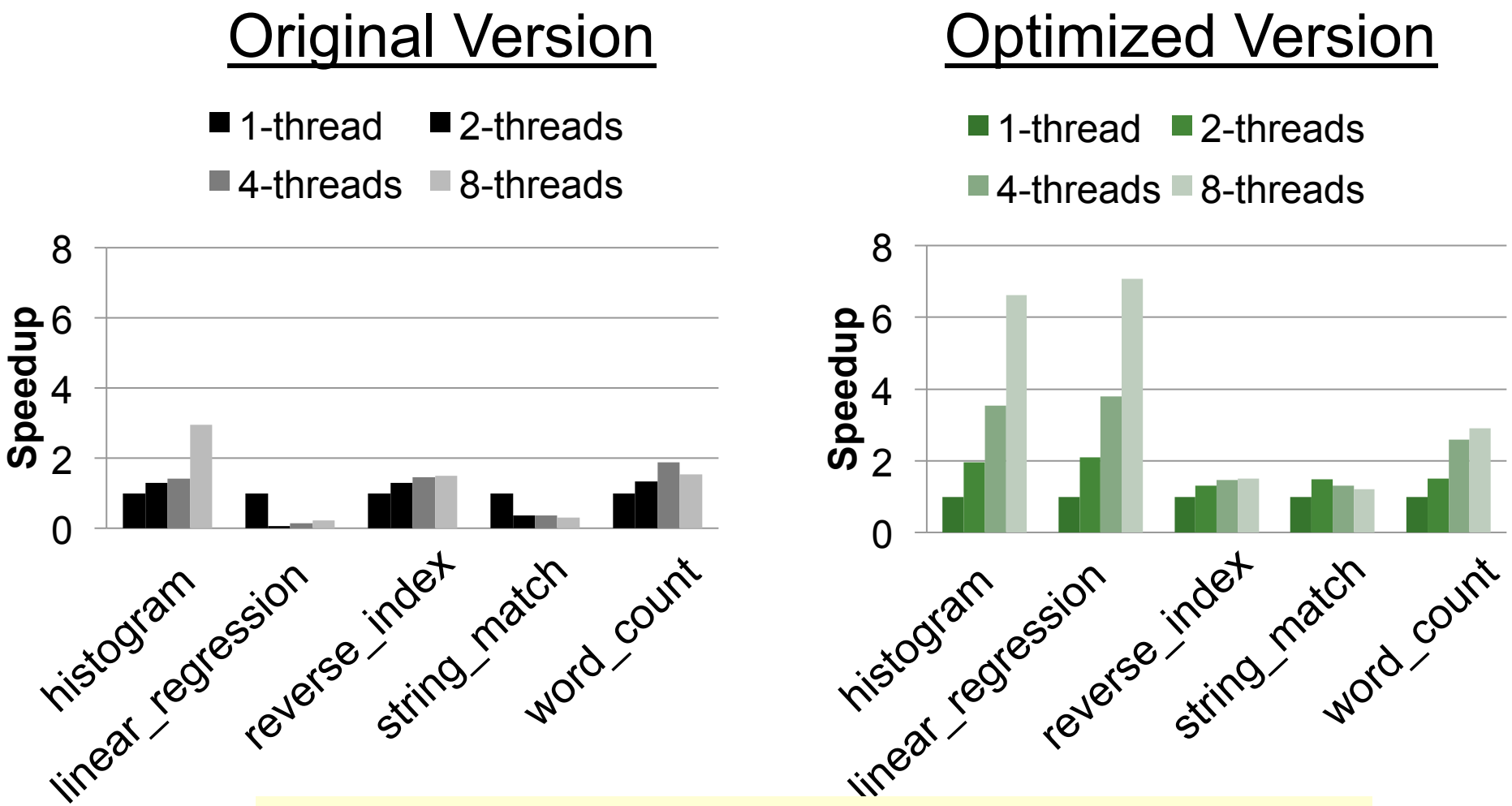


Array bounds used privately by threads were shared, stored in same cache line. Privatization led to 10X performance improvement; 30% for entire program



Optimized OpenMP version

Runtime False Sharing Detection



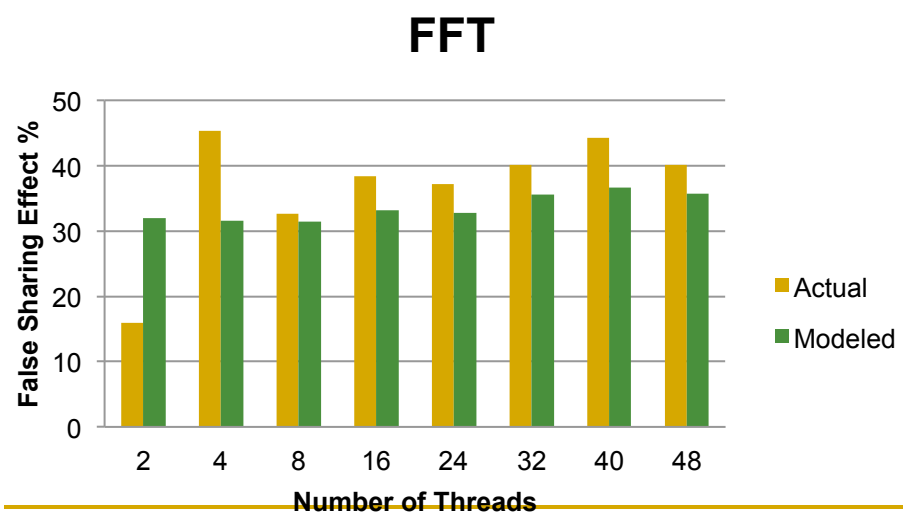
B. Wicaksono, M. Tolubaeva and B. Chapman. "Detecting false sharing in OpenMP applications using the DARWIN framework", LCPC 2011

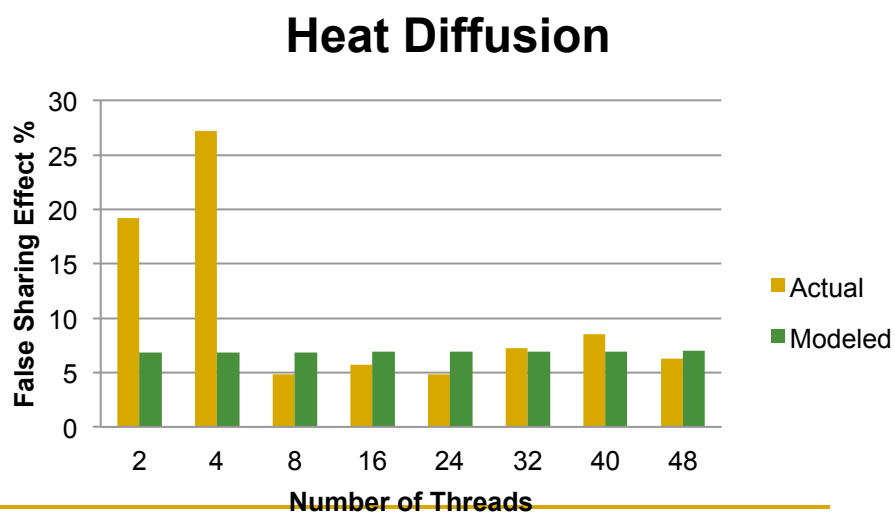
Modeling False Sharing at Compile-time

Compile-time assessment

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

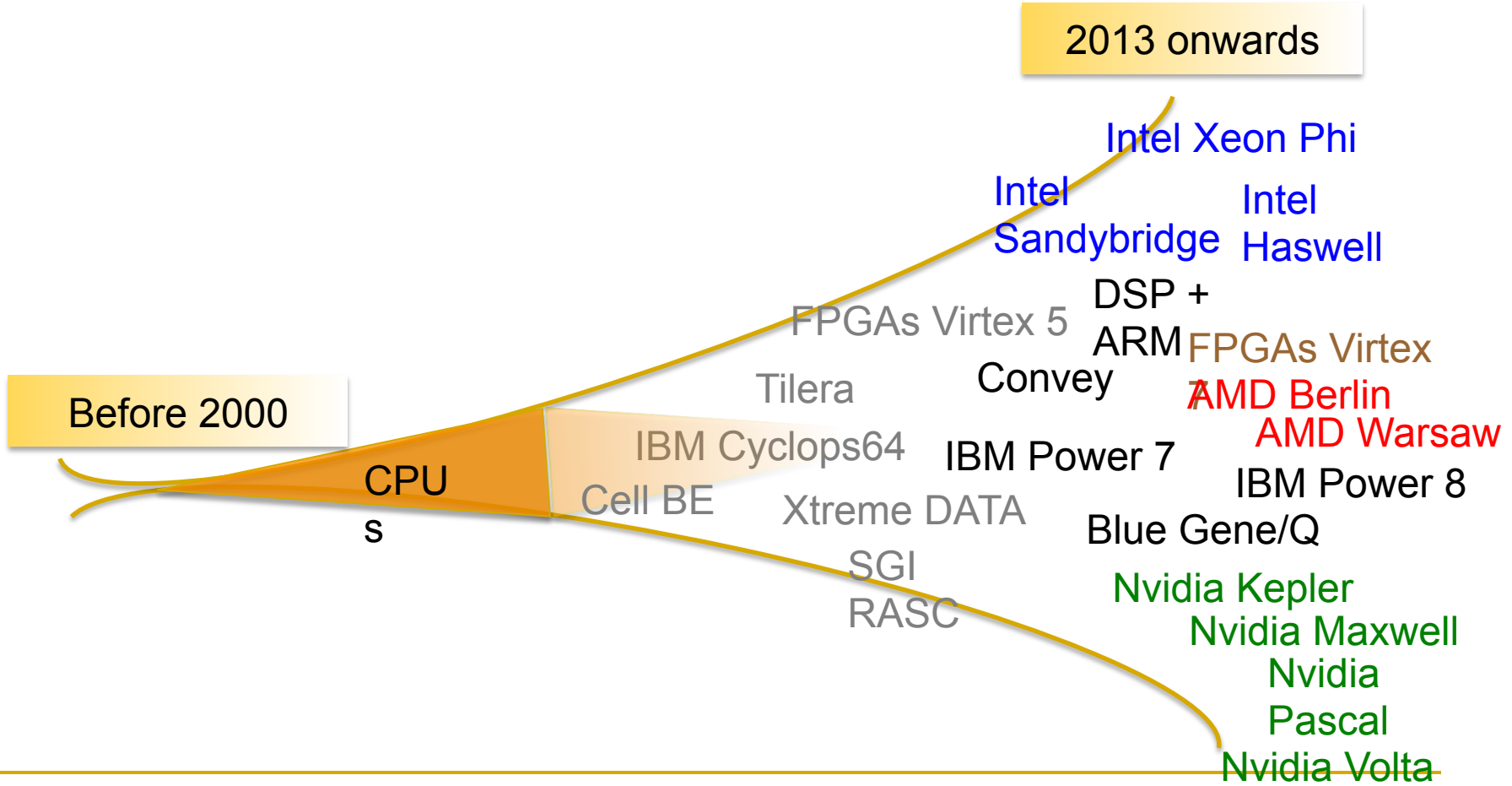
$$\frac{T_{fs\_measured} - T_{nfs\_measured}}{T_{fs\_measured}} \approx \frac{T_{fs\_modeled} - T_{nfs\_modeled}}{T^{*}_{fs\_modeled}}$$



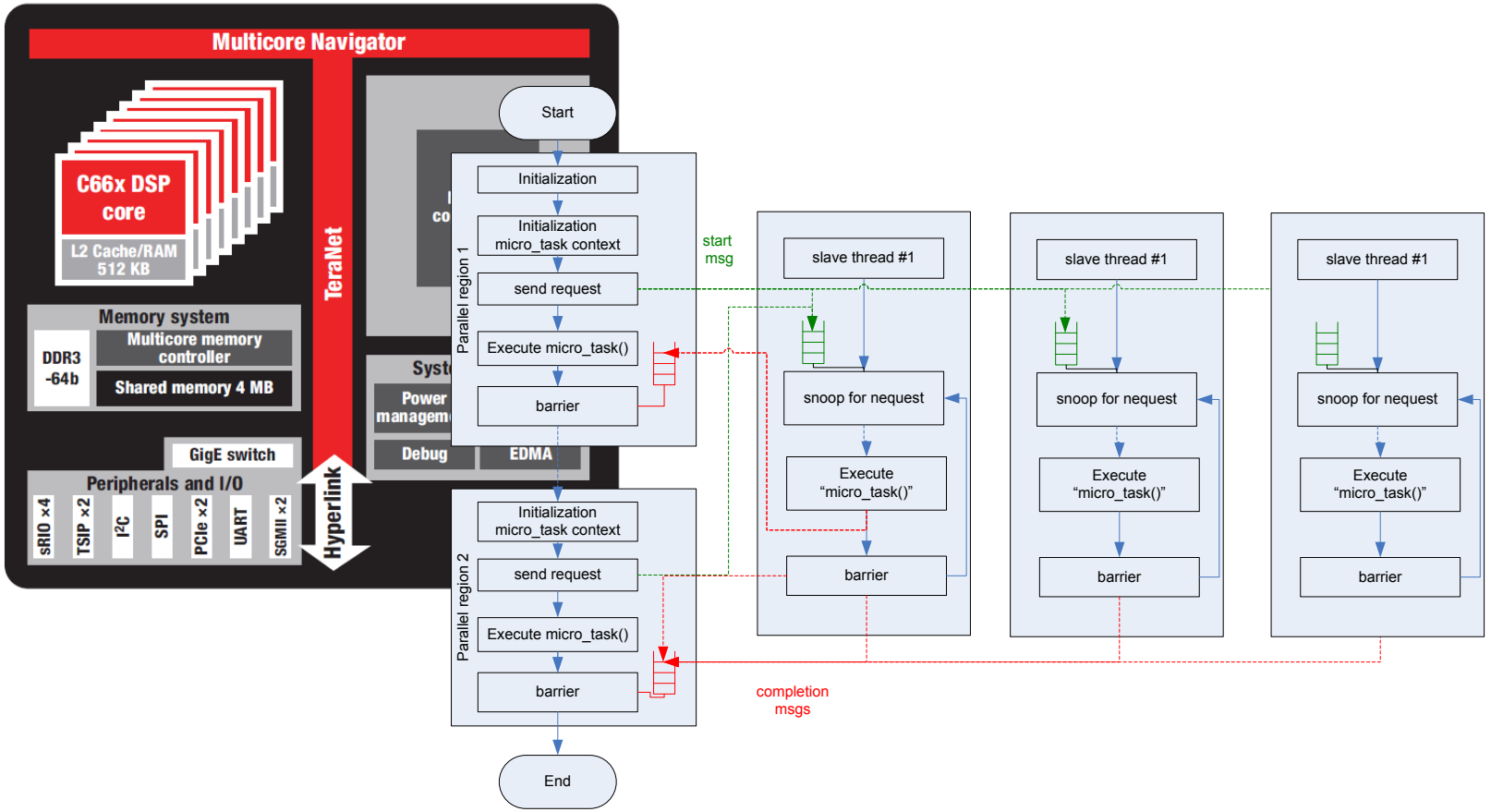


M. Tolubaeva, Y. Yan and B. Chapman. Compile-Time Detection of False Sharing via Loop Cost Modeling. HIPS'12 Workshop in conjunction with IPDPS'12 (accepted)

Heterogeneous doesn't just mean GPUs

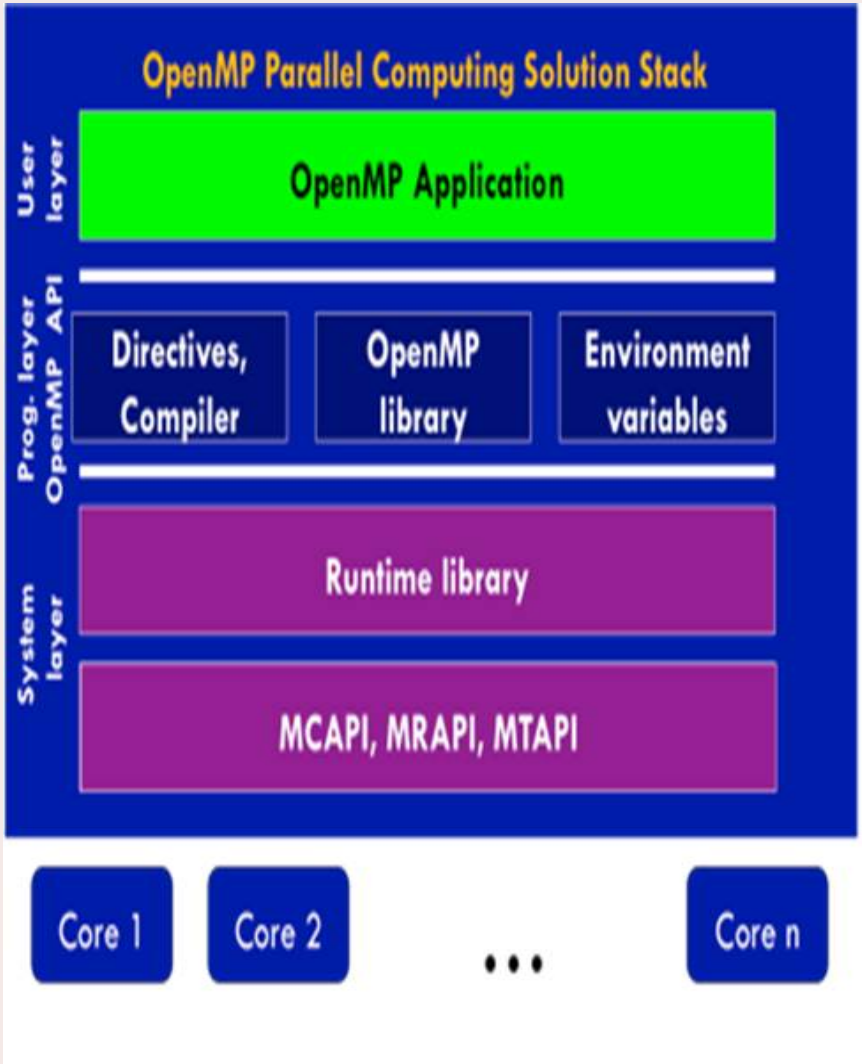


Rethinking Implementation Strategies



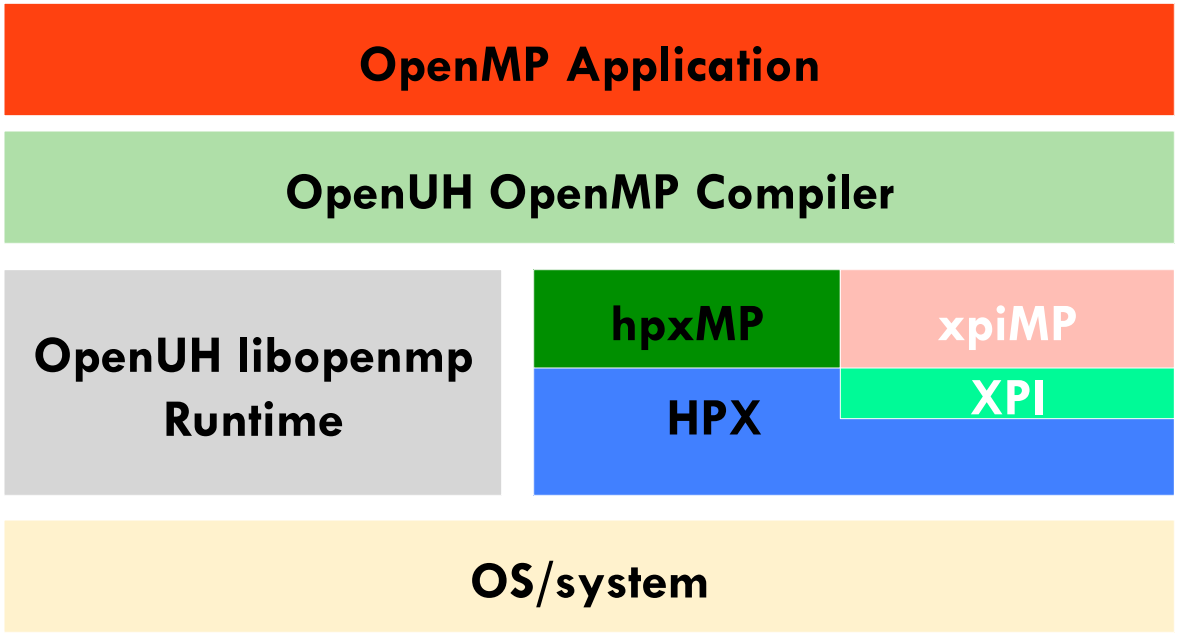
B. Chapman, L. Huang, E. Stotzer, E. Biscondi, A. Shrivastava, A. Gatherer. Implementing OpenMP on a High Performance Embedded Multicore MPSoC, pp 1-8, Proc. of Workshop on Multithreaded Architectures and Applications (MTAAP'09) In conjunction with International Parallel and Distributed Processing Symposium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

OpenMP for embedded systems solution stack



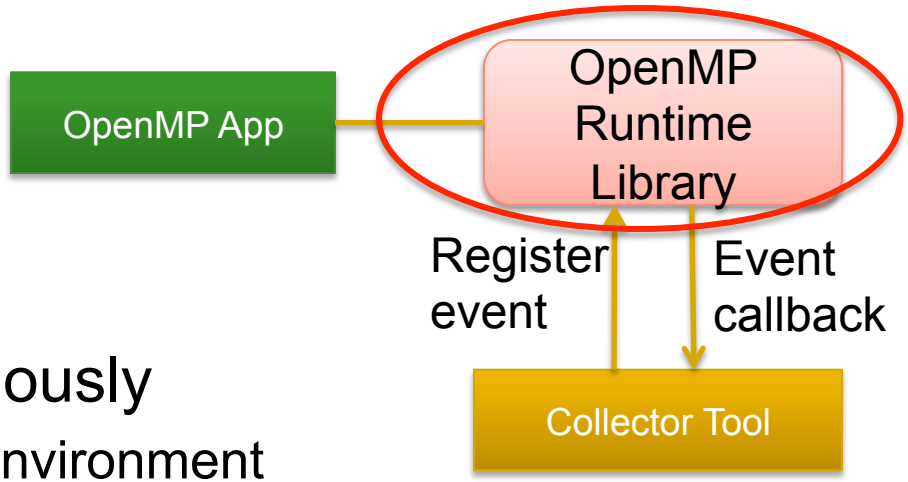
- Implementation of OpenMP for embedded systems
- Portable runtime layer supports implementation
- Using Multicore Association (MCA) APIs
  - Multicore Resource Management API
  - Multicore Communication API
  - Multicore Task Management API
  - Virtualization
- Very low overheads

OpenMP at Exascale: XPI as Runtime Target



- DOE-funded XPRESS project
  - Portable exascale system execution
  - Framework for future exascale system design
  - Experimentation with MPI and OpenMP on top of XPI

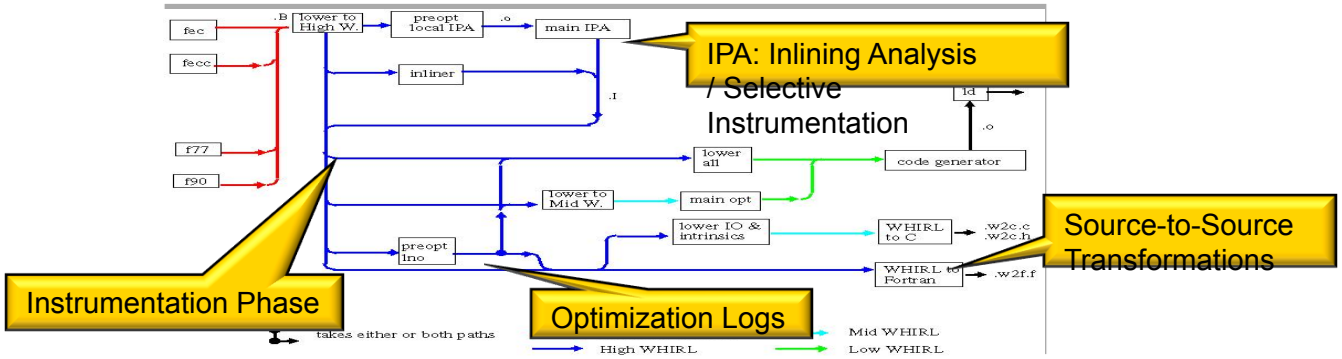
Adaptive Runtime



- 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
- Lightweight monitoring embedded compiler's runtime, enables monitoring of OpenMP program
  - Enables tools to interact with OpenMP runtime library
- Locality-aware scheduling
- Task-level autotuning, specific adaptations

Let's Change The System Stack!

- Compilers can share information on application with other components
- Facilitate lightweight data collection
- Multiversioning and runtime adaptation



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.