# Directive-Based Parallel Programming in an Age of Diversity

Barbara Chapman
Stony Brook University
University of Houston

Krakow, September 2015

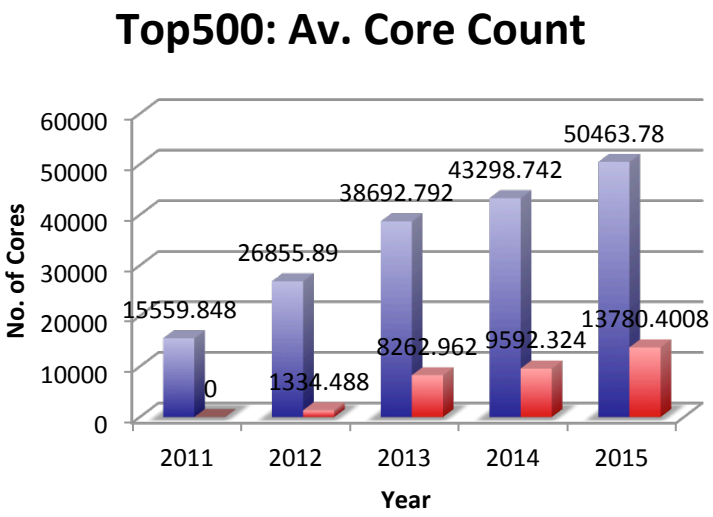


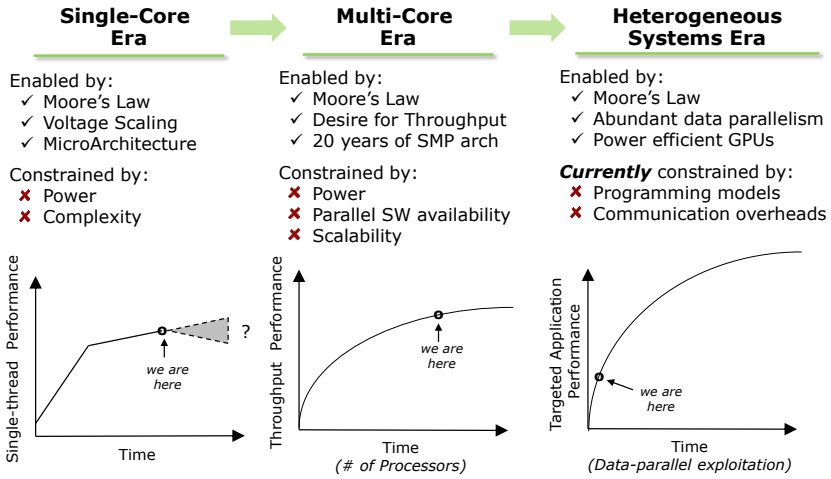
http://www.cs.uh.edu/~hpctools

# Agenda

- Changing High-Performance Computing (HPC) Architectures
- Implications for Application Development
- The Directive-Based Approach to Programming on the Node
- Some Possible Directions
- Tool Support for Porting Code

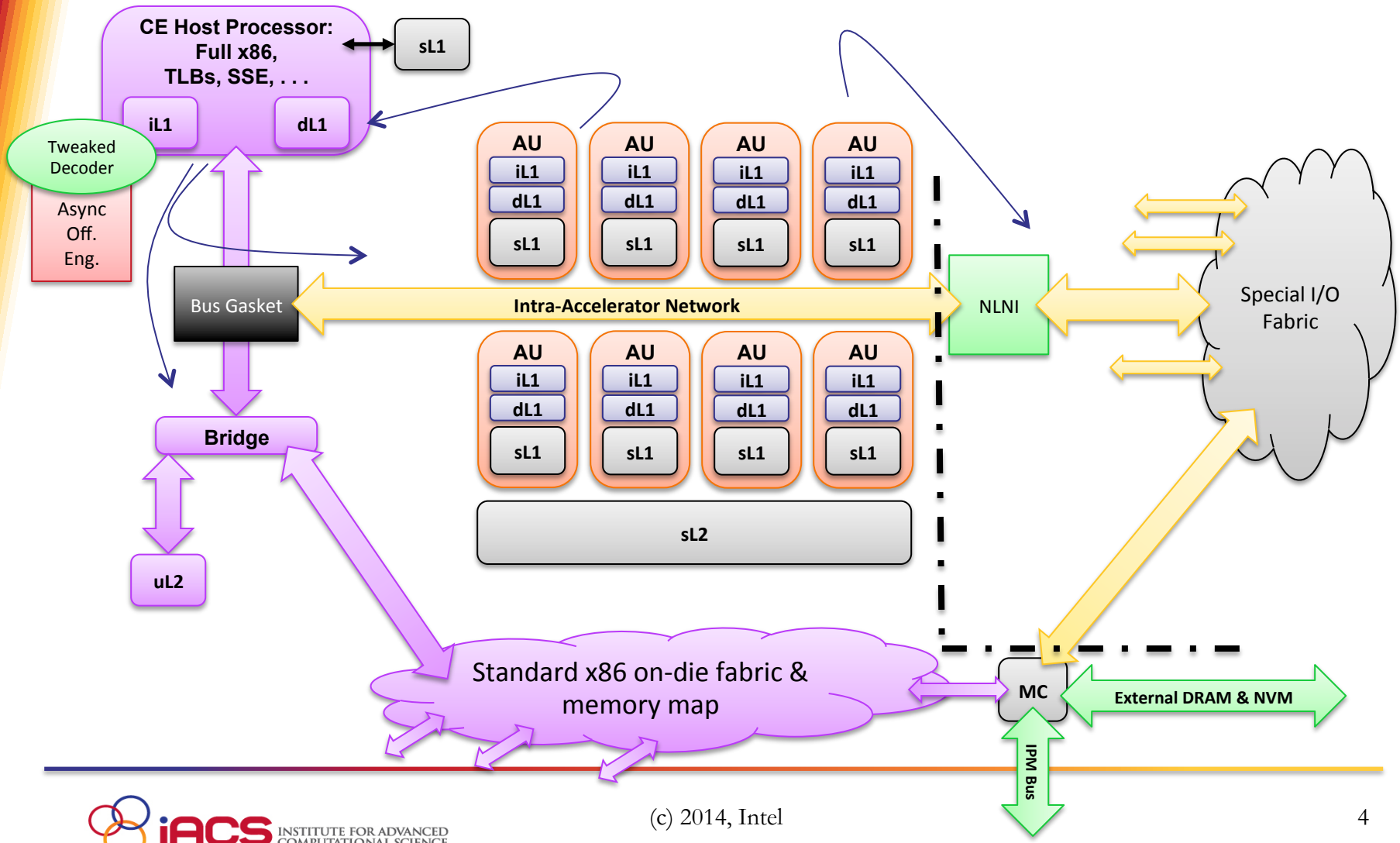
# On-Going Architectural Changes



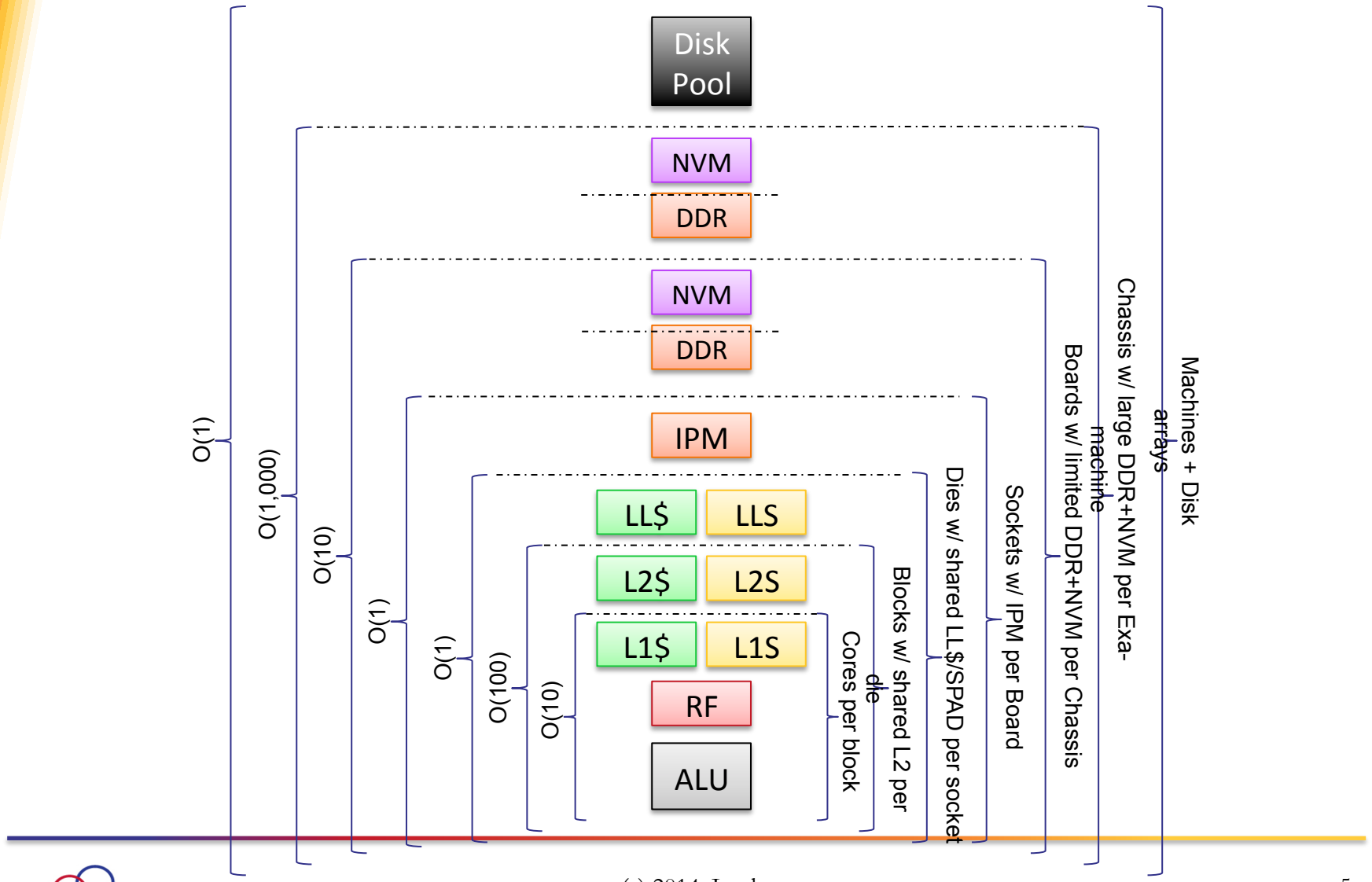




- Move to multi-/manycore nodes
  - Thermics, power are now key in design decisions
  - Massive increase in intra-node concurrency
  - Trend toward heterogeneity
  - Deeper, more complex memory hierarchies

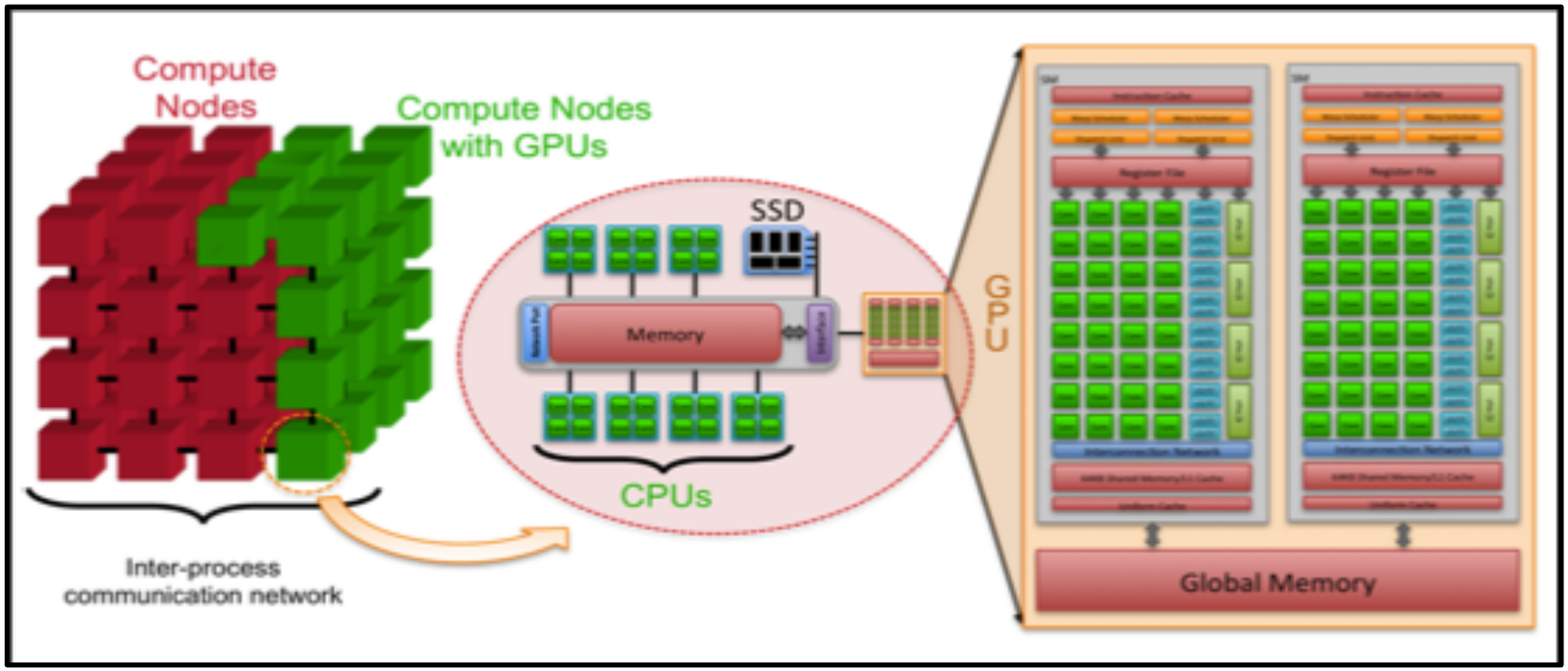
# Intel: "Sea of Blocks" Compute Model



# 10+ Levels Memory, O(100M) Cores



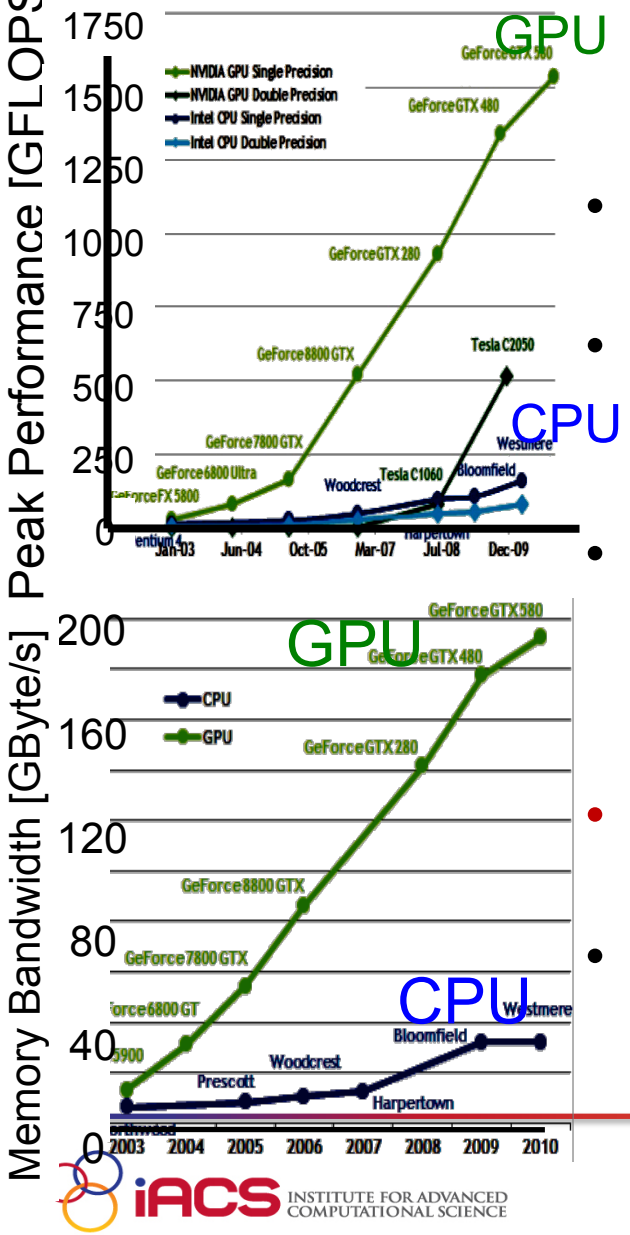
# Core Heterogeneity in HPC Systems




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

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

# TSUBAME2.0 GPU Rationalization

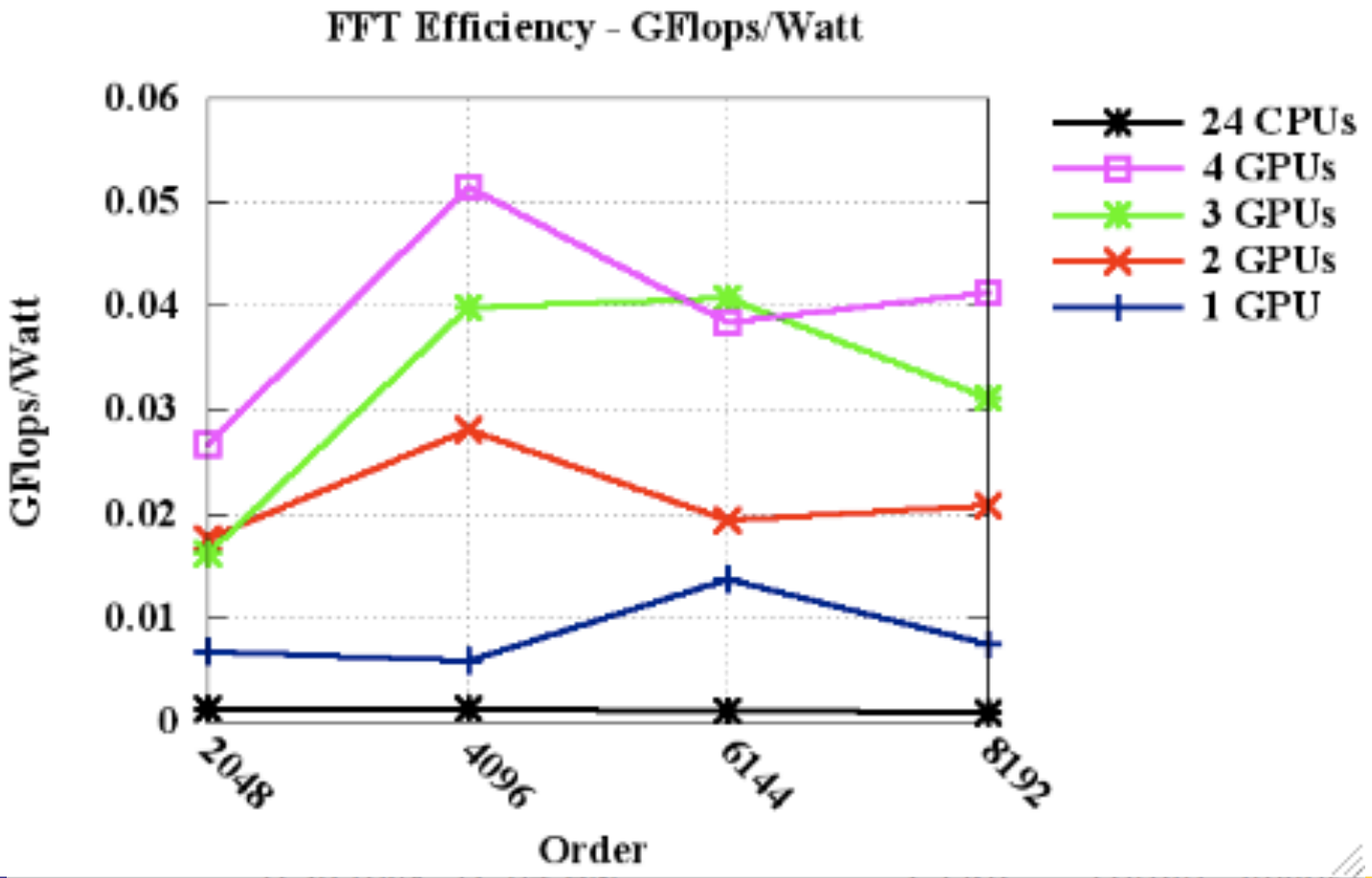



Courtesy: Satoshi Matsuoka, Tokyo Instutute of Technology

- ~3000 CPUs at 200+ Teraflops, ~4000 GPUs at 2.2 Petaflops
- Realistic best case: x5~6 perf gain per socket
  - Machine equivalent to 25,000~30,000 CPUs
- Alternative: CPU only, same $$$ and power, how big a system?
  - Answer: at best 5~6000 CPUs (Tsubame 1.0) at 400+ Teraflops
- **CPU equivalency = 1.4 x utilization x perf gain > 1.0 then we win!**
- No religious war but simple economics

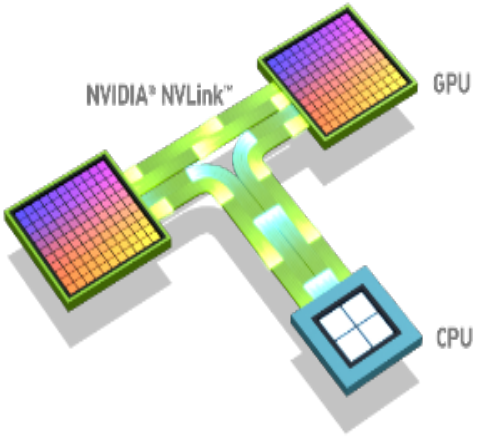

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

# Energy Efficiency: CPU vs GPU



# NVLink

## High-Speed GPU Interconnect




**KEPLER GPU**

**PASCAL GPU**

**Pascal**

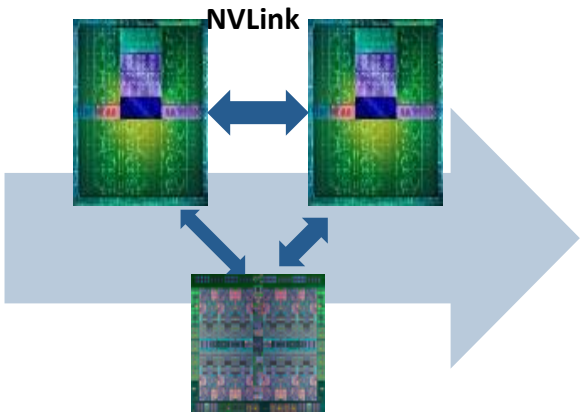
Unified Memory

3D Memory

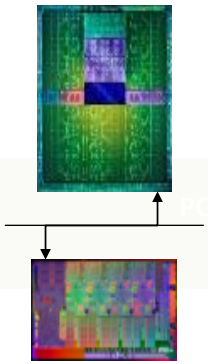
4x Higher Bandwidth (~1 TB/s)

3x Larger Capacity

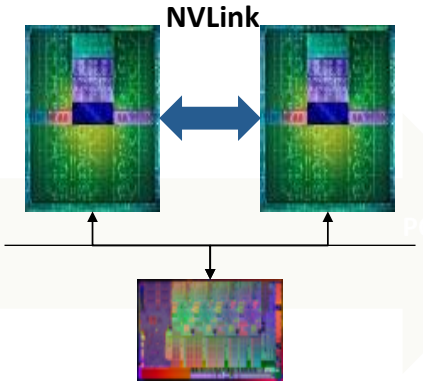
4x More Energy Efficient per bit




**POWER CPU**



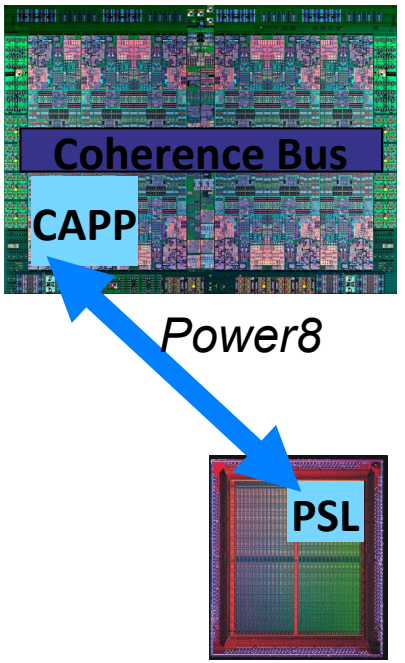

**X86, ARM64, POWER CPU**

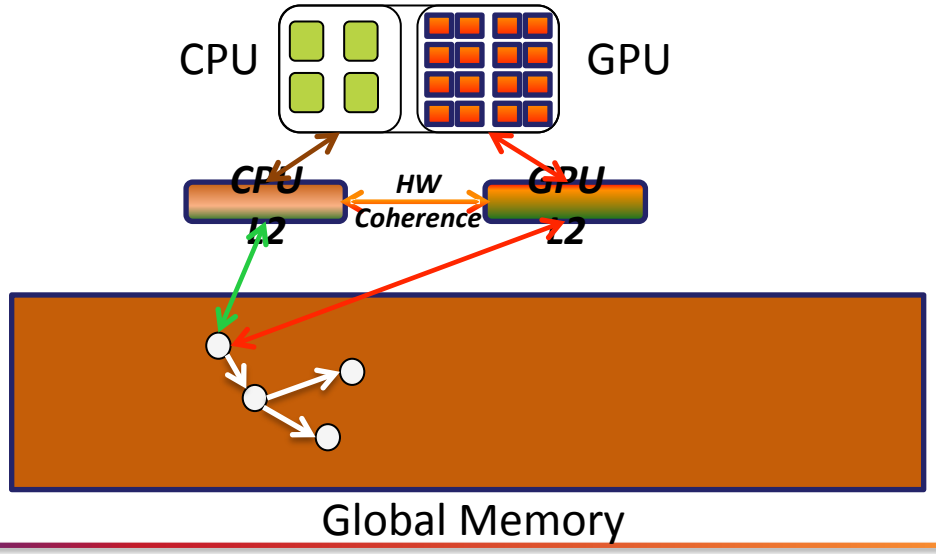



**X86, ARM64, POWER CPU**

# Integration of Accelerators: CAPI and APU

- IBM's Coherent Accelerator Processor Interface (CAPI) integrates accelerators into system architecture with standardized protocol
- Enables third parties to provide components
  - FPGAs, ASICs, …
- AMD's Heterogeneous System Architecture (HSA)-based APU also integrates accelerators



# Keystone II: 66AK2H12/06 SoC

## C66x Fixed or Floating Point DSP

- 4x/8x 66x DSP cores up to 1.4GHz
- 2x/4x Cotex ARM A15
- 1MB of local L2 cache RAM per C66 DSP core
- 4MB shared across all ARM

## Large on chip and off chip memory

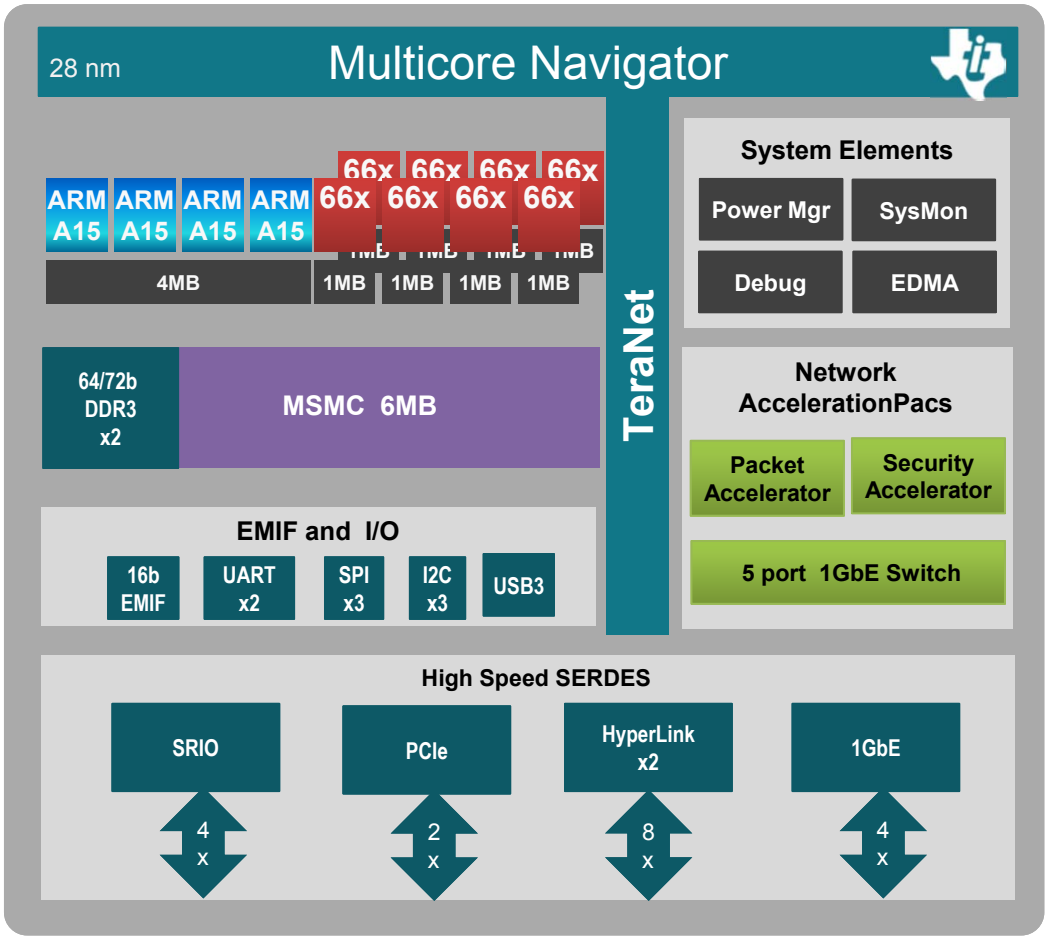
- Multicore Shared Memory Controller provides low latency & high bandwidth memory access
- 6MB Shared L2 on-chip
- 2 x 72 bit DDR3, 72-bit (with ECC), 10GB total addressable, DIMM support (4 ranks total)

## KeyStone multicore architecture and acceleration

- Multicore Navigator, TeraNet, HyperLink
- 1GbE Network coprocessor (IPv4/IPv6)
- Crypto Engine (IPSec, SRTP)

## Peripherals

- 4 Port 1G Layer 2 Ethernet Switch
- 2x PCIe, 1x4 SRIO 2.1, EMIF16, USB 3.0 UARTx2, SPI, I²C
- 15-25W depending upon DSP cores, speed, temp & other factors




40mm x 40mm package

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

# Agenda

- Changing High-Performance Computing (HPC) Architectures
- Implications for Application Development
- The Directive-Based Approach to Programming on the Node
- Some Possible Directions
- Tool Support for Porting Code

# HPC Applications: Requirements

- Performance
  - Must be able to exploit features of emerging machines at all levels
  - APIs must facilitate expression of concurrency, save power, use memory efficiently and exploit heterogeneity
- Performance portability
  - Implies not just that APIs are widely supported
  - But also that same code runs well everywhere
  - Very hard to accomplish

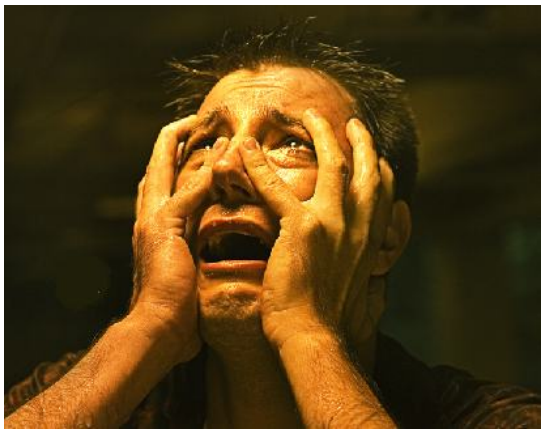
Performance less predictable in dynamic execution environment

# Developing HPC Applications

- Productivity
  - Need approaches that are reasonably easy to use
  - Hooks to get more performance where it is important
  - Need reasonable migration path for existing code
  - Along with interoperability to avoid unneeded rewrite
- Any new HPC programming languages out there?
  - Both MPI and OpenMP are being extended
  - New approaches may emerge, esp. task-based; DSLs will appear
  - Role of application developer in detecting/overcoming errors and efficient energy consumption not yet clear
- Libraries and directives are familiar approaches
  - Work under way to target MPI, OpenMP to proposed exascale runtime
  - Directive features may ultimately be integrated into base languages

# Productive Programming Models?

```
// Run one OpenMP thread per device per MPI node
#pragma omp parallel num_threads(devCount) if (initDevice())
 {
      // Block and grid dimensions
      dim3 dimBlock(12,12);
      kernel<<<1,dimBlock>>>();
      cudaThreadExit();
}
else
{
      printf("Device error on %s\n",processor_name);
}
      MPI_Finalize();
      return 0;
}
```



Code from *www.cse.buffalo.edu/faculty/miller/Courses/CSE710/heavner.pdf*

# Agenda

- Changing High-Performance Computing (HPC) Architectures
- Implications for Application Development
- The Directive-Based Approach to Programming on the Node
- Some Possible Directions
- Tool Support for Porting Code

www.openmp.org

# The OpenMP ARB 2015


- 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, ARM, Cray, Fujitsu, HP, IBM, Intel, Micron, NEC, Nvidia, Oracle, Red Hat, Texas Instruments
    - Auxiliary: ANL, ASC/LLNL, BSC, cOMPunity, EPCC, LANL, LBNL, NASA, ORNL, RWTH Aachen, SNL, TACC, University of Houston
- **Attend IWOMP and OpenMPCon**: http://www.iwomp.org

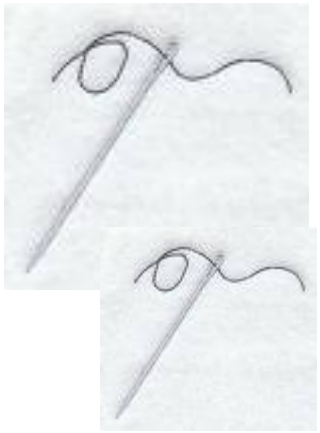
"High-level directive-based multi-language parallelism that is performant, productive and portable"

# Use of OpenMP

- Moderate-size scientific, technical applications
  - Initially, Fortran binding only
- General-purpose multicore programming
  - Tasks, C and C++ bindings
- Embedded systems
  - Tasks, kernel offloads
- Large-scale parallel computations
  - Usually, in conjunction with MPI
- Entry-level parallel programmers

Many application developers think that the use of directives means they don't have to restructure code. That is not true.

# OpenMP



- Oct 1997 – 1.0 Fortran
- Oct 1998 – 1.0 C/C++
- Nov1999 – 1.1 Fortran: interpretations added
- Nov 2000 – 2.0 Fortran (F95, nested locks)
- Mar 2002 – 2.0 C/C++
- May 2005 – 2.5 Fortran/C/C++ (one API, multiple bindings, memory model, ICVs, terminology)
- May 2008 – 3.0 (task execution model, explicit tasks, parallelization of multiple loop levels, nested parallelism; wait policy)
- July 2011 - 3.1 (final, mergeable tasks, taskyield, atomic construct)
- July 2013 – 4.0 (support for devices, target and data mapping; SIMD loops; thread affinity; task dependences; user defined reductions)

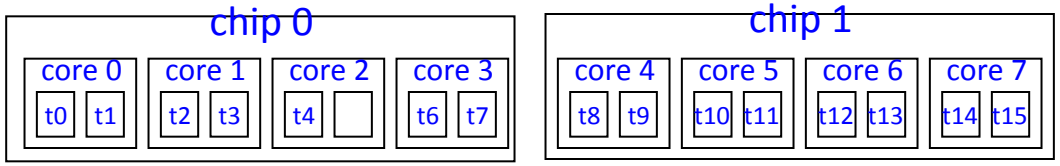
Runtime routines: 10 in 1.1; 19 in 3.0; 28 in 4.0

# 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

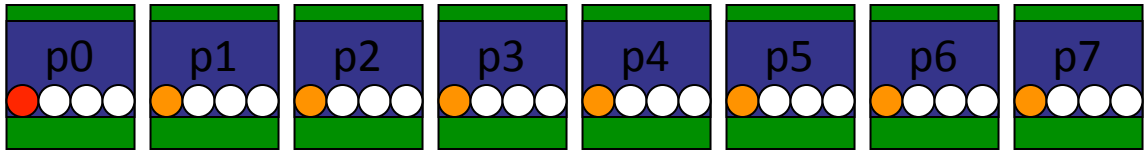
```
#pragma omp parallel
#pragma omp for schedule(dynamic)
      for (I=0;I<N;I++){
            NEAT_STUFF(I);
      } /* implicit barrier here */
```

# OpenMP 4.0 Affinity



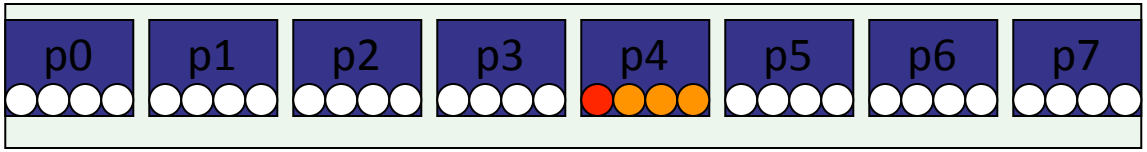

- OpenMP Places and thread affinity policies
  - **`OMP_PLACES`** to describe hardware regions
  - **`affinity(spread|compact|true|false)`**
- **`SPREAD`**: spread threads evenly among the places

  **`spread 8`**

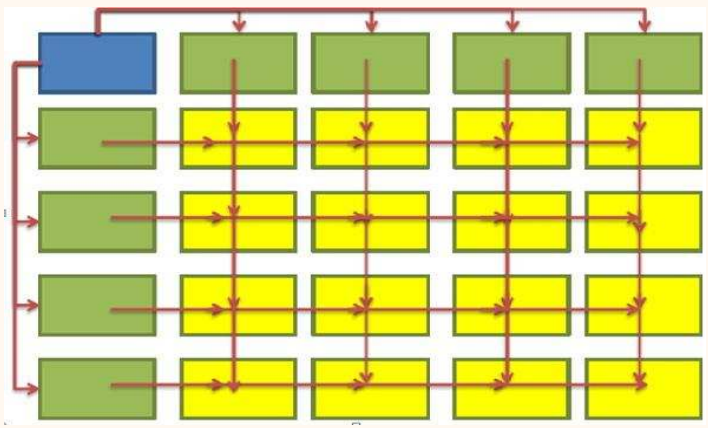



- **`COMPACT`**: collocate OpenMP thread with master thread

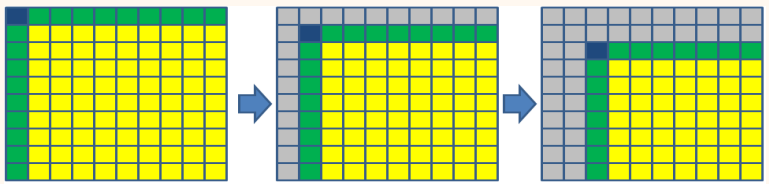
  **`compact 4`**

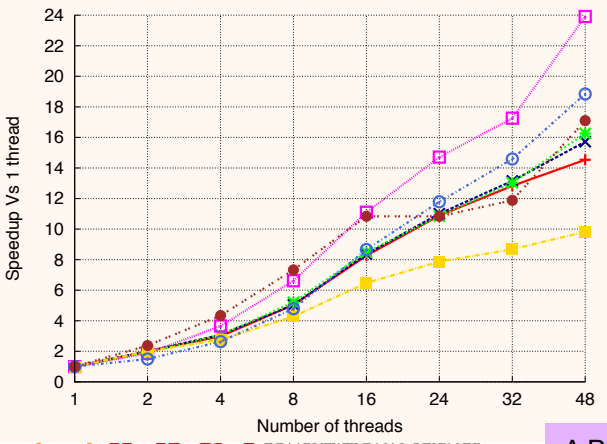


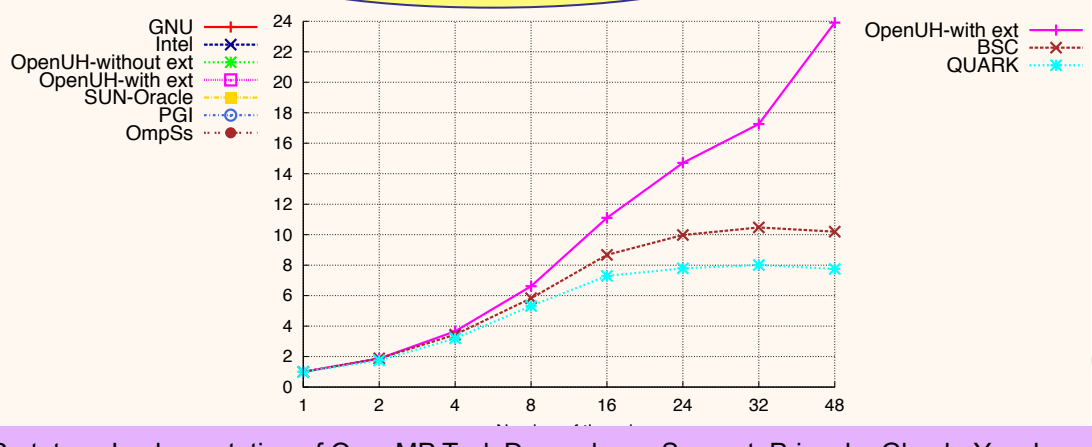
# Dependent Asynchronous Tasks



```
1  #pragma omp parallel
2  {
3   #pragma omp master
4   {
5     for ( i=0; i<matrix_size; i++ ) {
6
7     /**** Processing Diagonal block ****/
8     ProcessDiagonalBlock (.....);
9
10    for (i=1;i<M;i++){
11
12    #pragma omp task out(2*i) /** Processing block on column **/
13      ProcessBlockOnColumn (.....);
14
15    #pragma omp task out(2*i+1) /** Processing block on row **/
16      ProcessBlockOnRow (.....);
17      }
18
19  /*** Elimination of Global Synchronization point *********/
20
21  /**** Processing remaining inner block ****/
22    for (i=1;i<M;i++)
23      for (j=1;j<M;j++){
24    #pragma omp task in(2*i) in(2*j+1)
25    ProcessInnerBlock (.....);
```





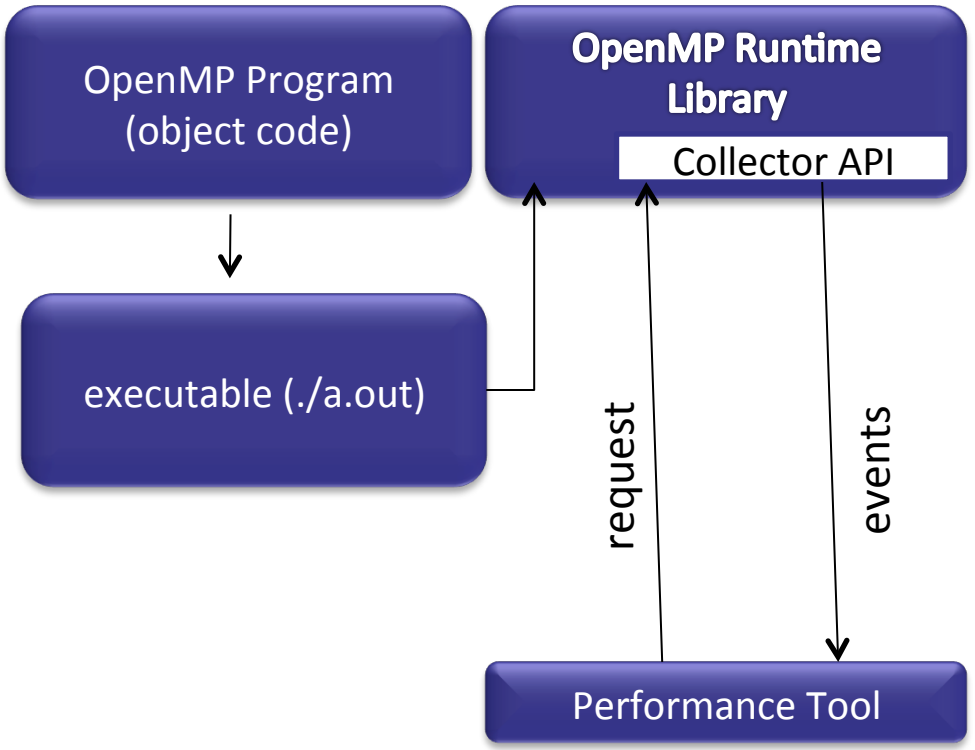






A Prototype Implementation of OpenMP Task Dependency Support; Priyanka Ghosh, Yonghong Yan, Deepak Eachempati and Barbara Chapman; International Workshop on OpenMP (IWOMP) 2013

# OpenMP Performance Tools Interface

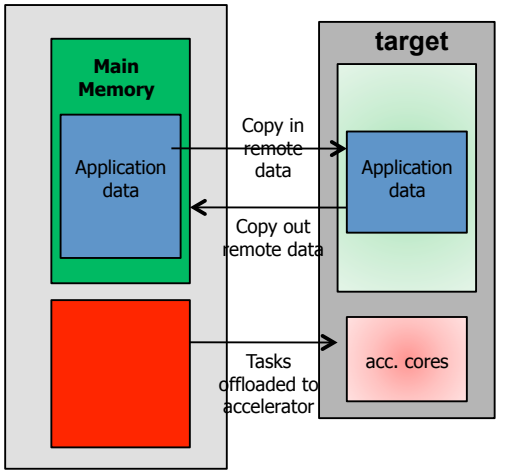
- A single routine, used by tools to communicate with runtime
- `int __omp_collector_api(void *msg)`
- Designed to support events/states needed for statistical profiling and tracing tools
- Extends original design from Sun Microsystems (Collector Interface)

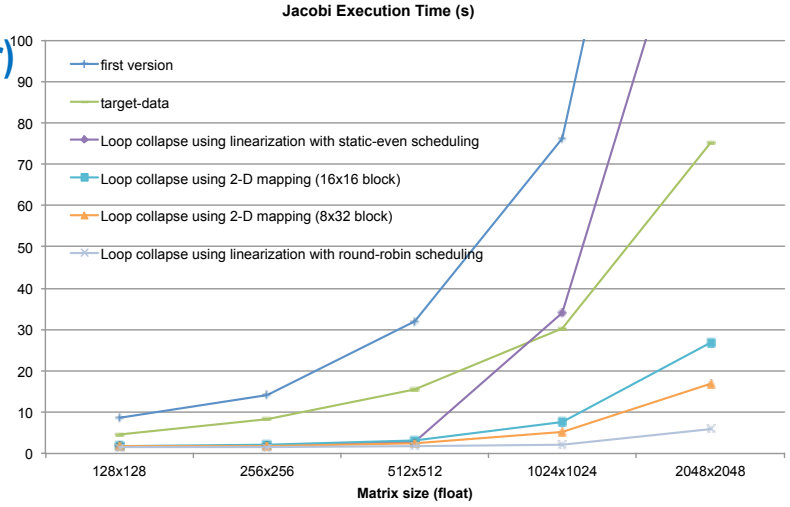


# OpenMP for Accelerators

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

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



# Looking Ahead: OpenMP 4.1

- Device construct enhancements
  - more control, flexibility in data movement between host and devices
  - asynchronous support with **nowait** and **depends**
  - multiple device types
  - "deep copy" for pointer-based structures/objects
- Loop parallelism enhancements
  - extended **ordered** clause to support *do-across* (e.g. wavefront) parallelism for loop nests
  - new **taskloop** construct for asynchronous loop parallelism with control over task grain size
- Array reductions for C and C++
- Under consideration:
  - memory affinity
  - task priorities (very likely)
- and more!



Draft of 4.1 already available at www.openmp.org

# Further Ahead: OpenMP 5.0

Many features under consideration:

- Better device support
- Interoperability and composability
- Locality and affinity
- General error model
- Transactional memory
- Additional looping constructs
- Recent C/C++ standards
- Enhanced tasking support (tasks outside parallel regions?)

# 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
- Compilers: PGI, Cray, CAPS, OpenARC, OpenUH, GCC (4.9)
- Attend OpenACC workshop, 12. October ( www.openacc.com)

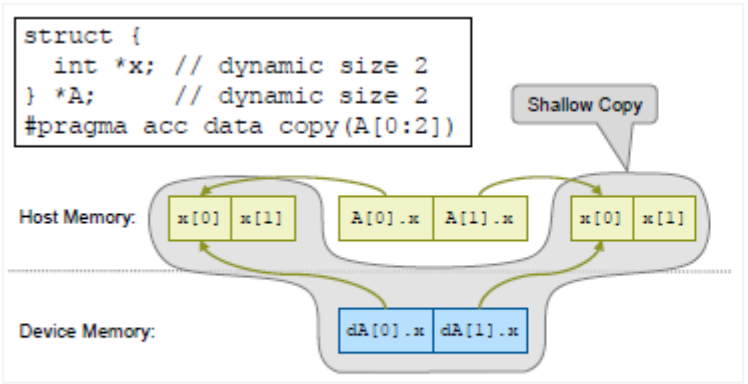
# OpenACC Features

- High-level directive-based programming API for accelerators such as GPUs, APUs, Intel's Xeon Phi, FPGAs and even DSP chips.
- Data directives: *copy, copyin, copyout*, etc
- Data synchronization directive: *update*
- Compute directives
  - *parallel*: more control to the user
  - *kernels*: more freedom to the compiler
- Three levels of parallelism: gang, worker and vector
- Commercial OpenACC compilers
  - PGI, CRAY, PathScale
- Open source OpenACC compilers
  - GCC 5.0, OpenARC, OpenUH, RoseACC, etc.

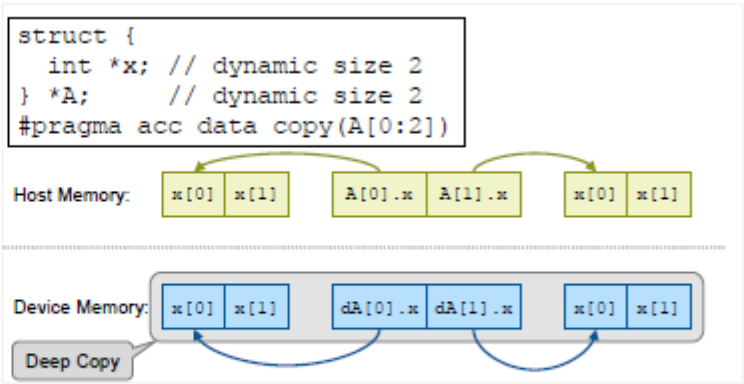
# OpenACC Status

- Current Status
  - 1.0: structured data region, computation offloading
  - 2.0: unstructured data region, nested parallelism
  - 2.5 (draft): OpenACC profiling interface
- Work in progress: 3.0 and later
  - Data deep copy
  - Multithreading and OpenACC
  - Multiple devices, homogeneous and heterogeneous
  - Multiple devices as a single virtual device
  - Host as a device

# Complex Data Management in OpenACC




(a) Shallow copy




(b) Deep copy

- Simple, elegant solution to deep copy problem is a major challenge
- Proposal adds **shape** and **policy** directives
- Directive-based specification requires no modification to underlying code

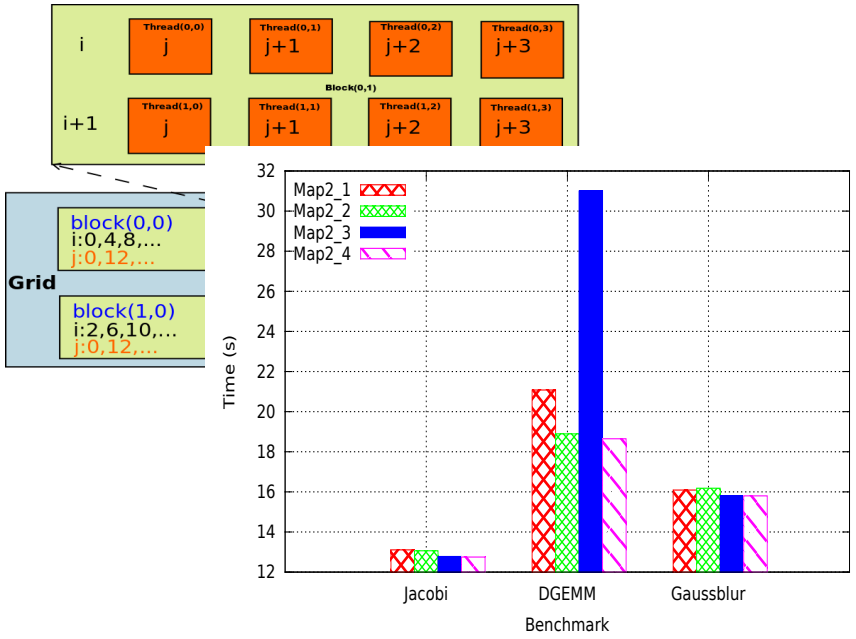


Figure from TR-14-1, **Complex Data Management in OpenACC™ Programs**

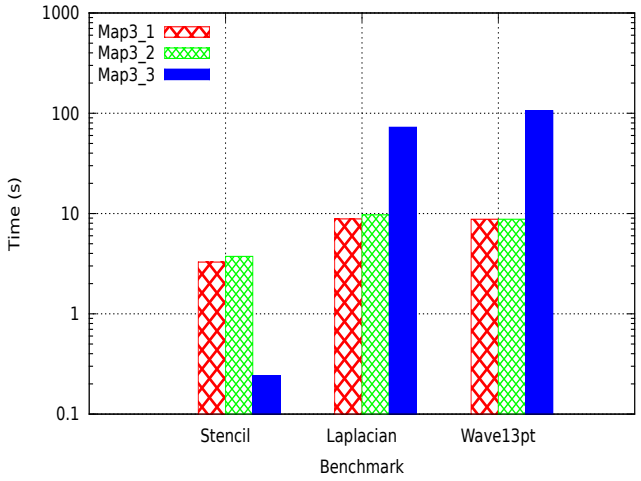
# OpenACC Compiler Translation

- Need to achieve coalesced memory access on GPUs

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




Double nested loop mapping.



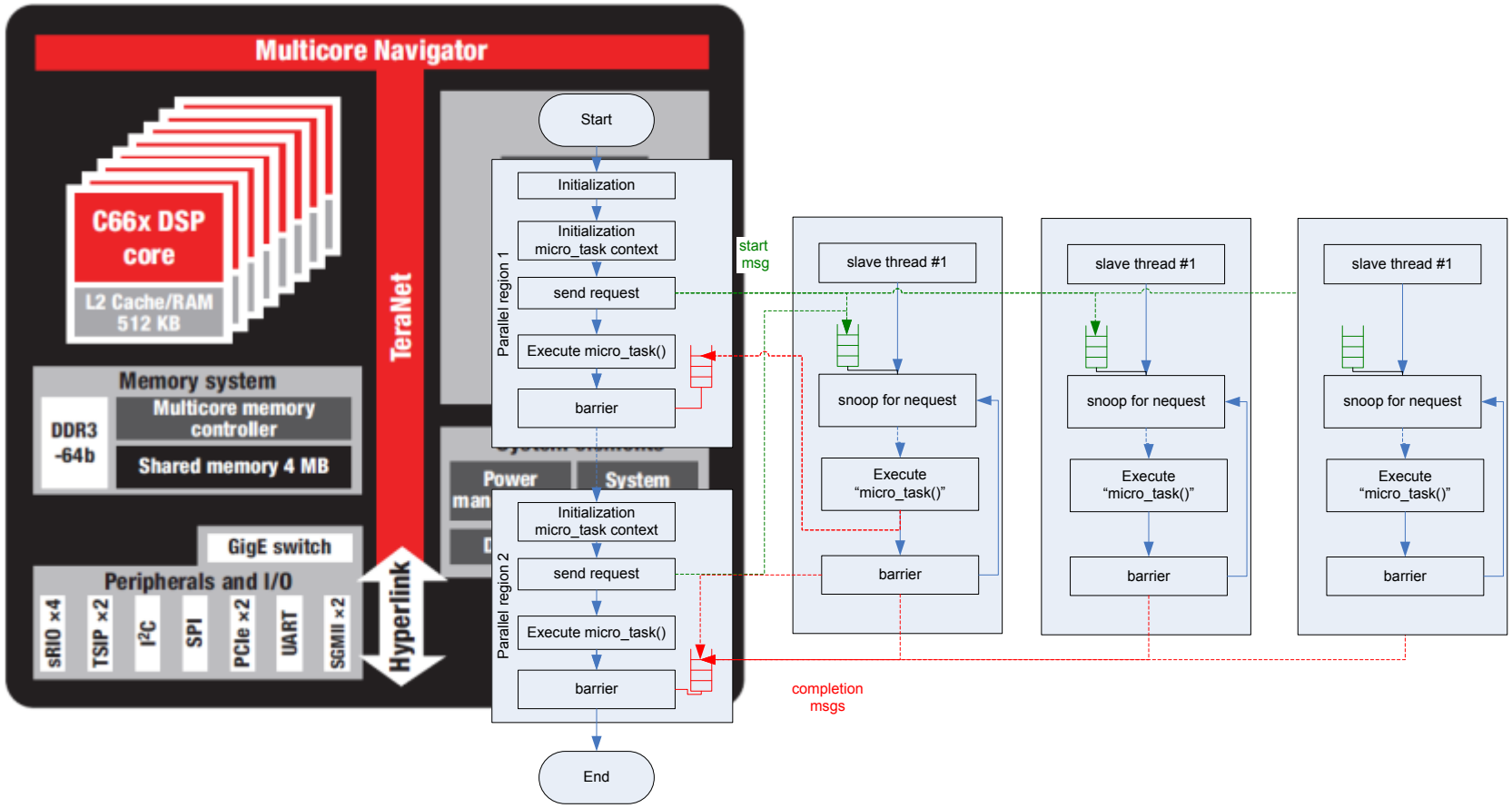

Triple nested loop mapping.

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

# Agenda

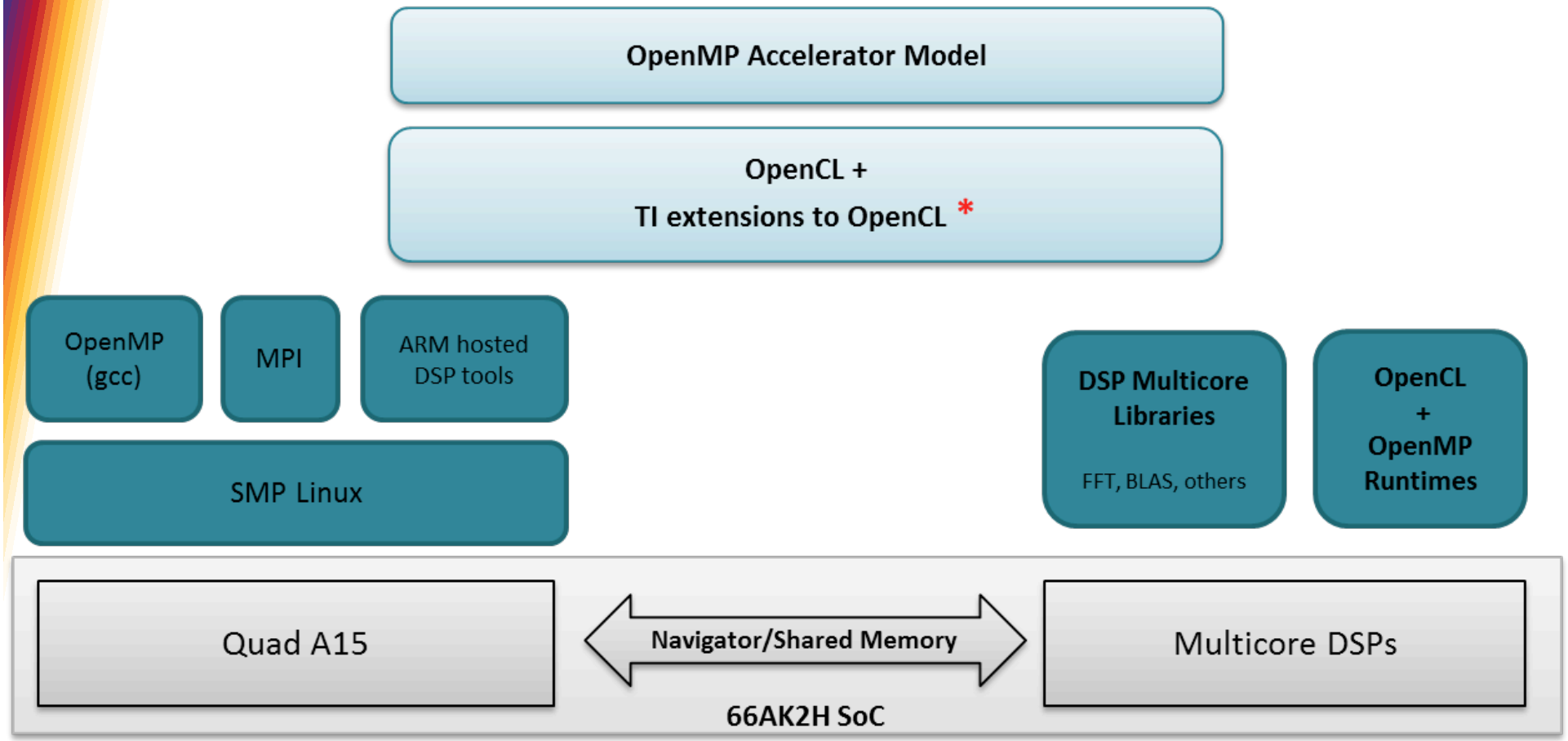
- Changing High-Performance Computing (HPC) Architectures
- Implications for Application Development
- The Directive-Based Approach to Programming on the Node
- Some Possible Directions
- Tool Support for Porting Code

# OpenMP on Low-Power Architecture, 2009




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.

# Programming Model for Keystone

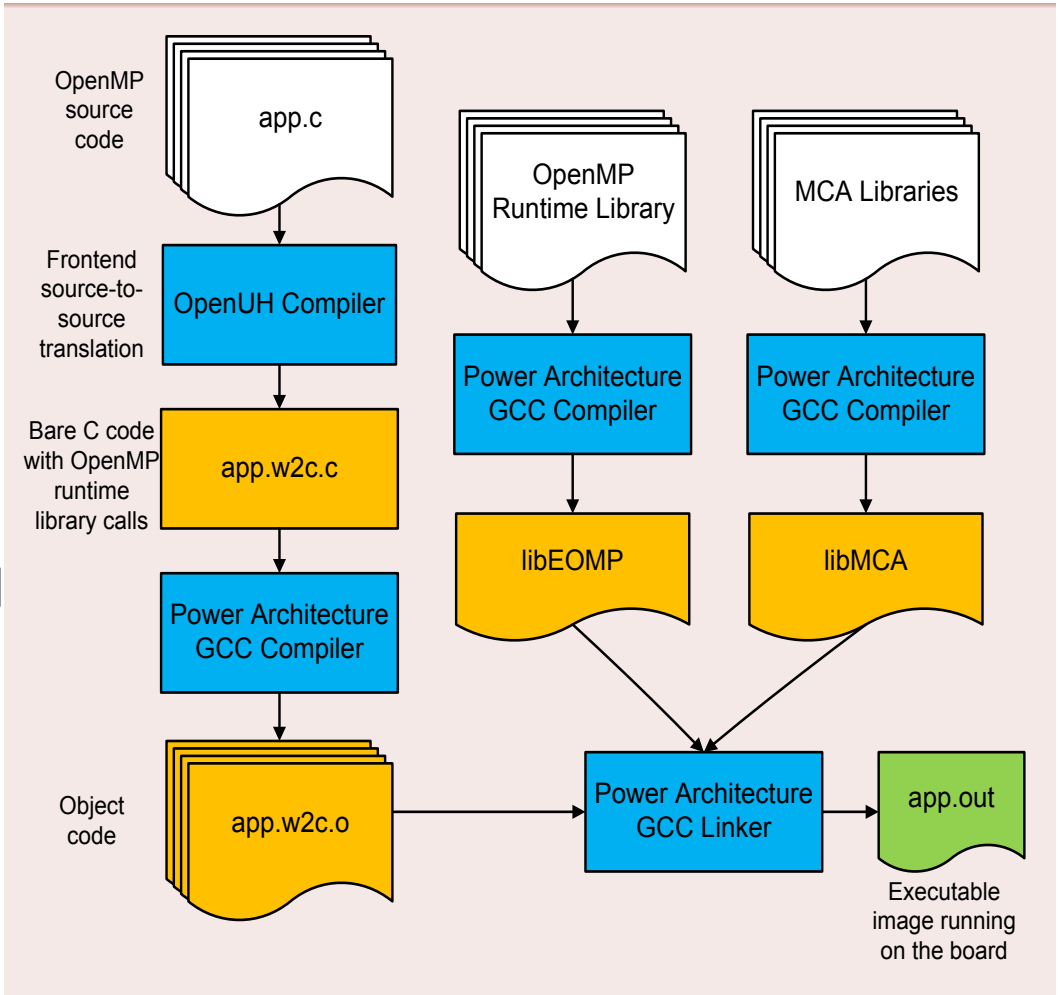



* TI extensions enable OpenCL kernels to act as wrappers for C code with OpenMP regions

# OpenMP+MCA Libraries on Freescale DSP, 2014

- Use MxAPIs as the translation layer for OpenMP
- Translate C+OpenMP to C with runtime function calls
- PowerPC-GCC as the back-end to generate object files and libraries
- Final executable file is generated by linking the object file, our OpenMP runtime library and the MCA runtime library

*(Papers at LCTES, PMAM, Talks at SIAM, SRC Review Meeting, Articles in EE times)*



# Task-Based OpenMP Execution, 2002

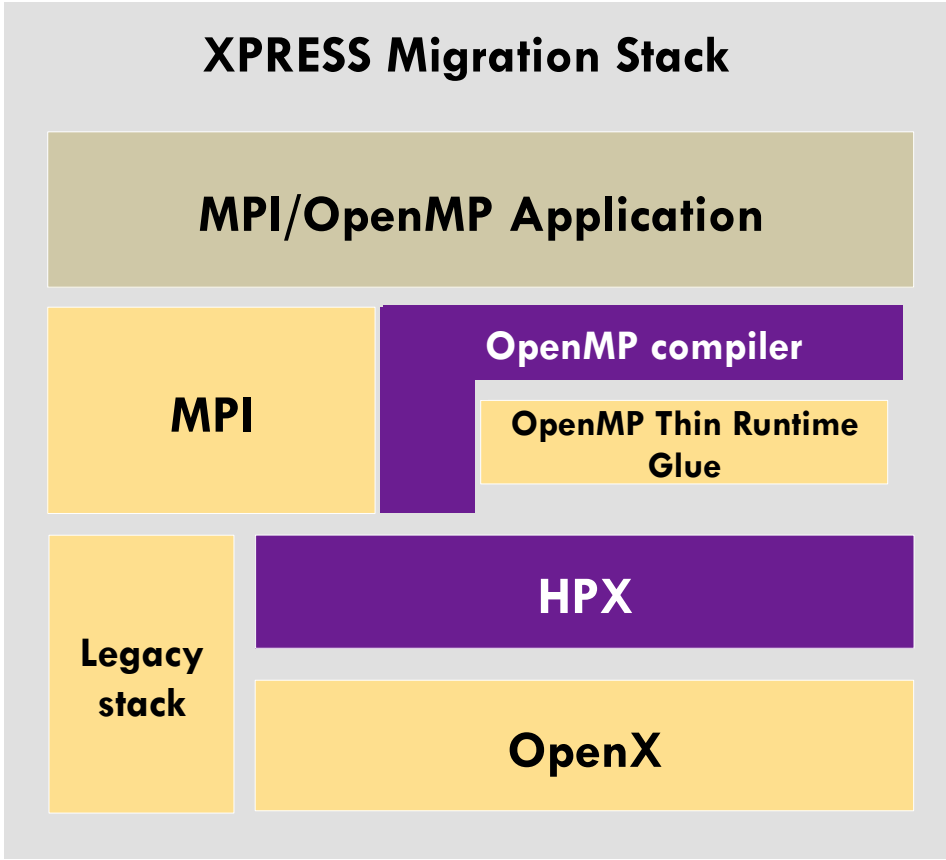
- Compiler translates "standard" OpenMP into collection of tasks and task graph
  - Analyzes data usage per task
- What is "right" size of task?
  - Might need to adjust at run time
- Runtime trade-off between load balance, co-mapping of tasks that use same data
- In-situ mappings work best: **execute task where data is**



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

# OpenMP in an Exascale World

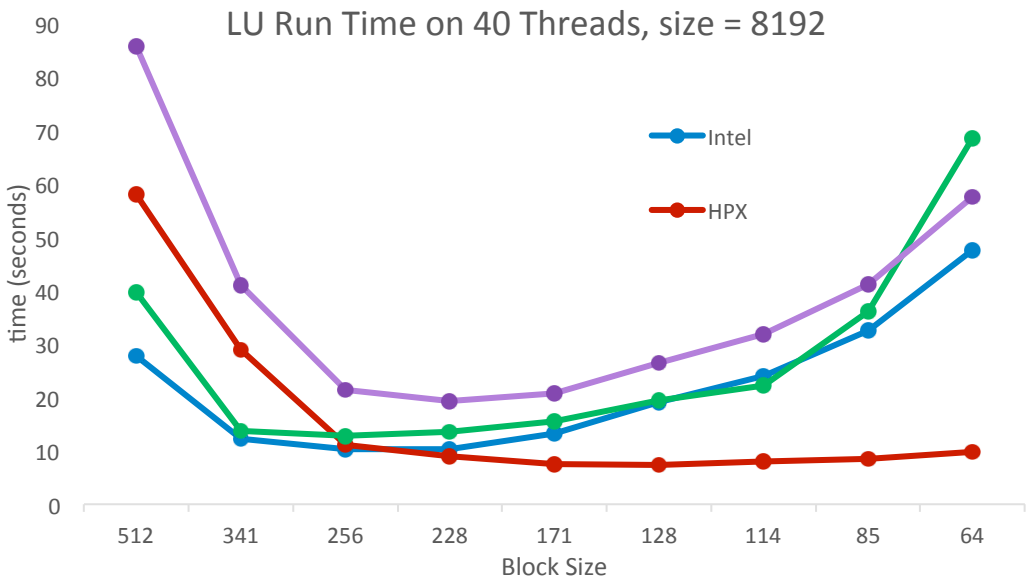
- *OpenX: prototype* software stack for Exascale systems
  - HPX is runtime system
  - Lightweight threads
  - Thread migration for load balancing, throughput.
- Translating OpenMP -> HPX
  - Maps OpenMP task and data parallelism onto HPX
  - Exploit data flow execution capabilities at scale
  - Big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



# OpenMP over HPX (on-going work)

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

- **OpenMP translation:**
  No direct interface to OS threads
  - No tied tasks; threadprivate tricky, slow
  - Doesn't support places, private memory
  - OpenMP task dependencies via futures
  - HPX locks faster than OS locks



# HPC Meets Big Data

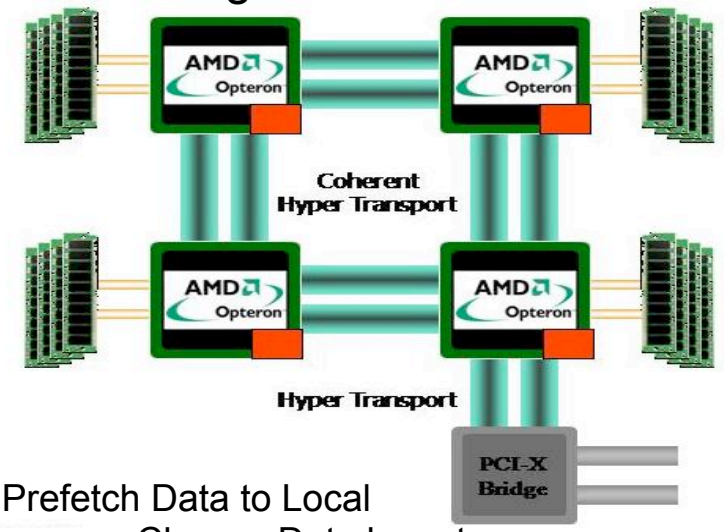
- HPC vs. Big Data
  - Flops vs. throughput
  - Scientific computation vs. data analytics (machine learning)
  - MPI+OpenMP vs. Hadoop/Spark/Cassandra…
  - Big Data developers expect simple programming interfaces
- A productive programming environment for both?
  - Accelerated processing for highly responsive applications
  - Optimized data transfers using HPC techniques; minimized data movement across entire application
  - Execution scheduler to optimize response time, adapt
- Can HPC-like directives help real-time processing of big data?
  - Increase scope of big data computations
  - Along with relative ease of application development

# Agenda

- Changing High-Performance Computing (HPC) Architectures
- Implications for Application Development
- The Directive-Based Approach to Programming on the Node
- Some Possible Directions
- Tool Support for Porting Code

# Code Restructuring?

Heterogeneous node:




T5: Data distribution for elem(ie) (GPU/CPU memory)

```
1: subroutine compute_and_apply_rhs(np1,nm1,n0,dt2,elem,hvcoord,hybrid,&
2:       deriv,nets,nete,compute_diagnostics)
3: ...
4: do ie=nets,nete
5:   do q=1,qsize
6:     if (tracer_advection_formulation==TRACERADV_TOTAL_DIVERGENCE) then
7:          do k=1,nlev
8:             gradQ(:,:,1)=elem(ie)%state%v(:,:,1,k,n0)*elem(ie)%state%Qdp(:,:,k,q,n0)
9:             gradQ(:,:,2)=elem(ie)%state%v(:,:,2,k,n0)*elem(ie)%state%Qdp(:,:,k,q,n0)
10:            divdp = divergence_sphere(gradQ,deriv,elem(ie))
11:            do j=1,nv
12:               do i=1,nv
13:                  qtens(i,j,k,q)=-divdp(i,j)
14:               enddo
15:            enddo
16:         enddo
17:     else
18: .....
19: call bndry_exchangeV(hybrid,edgeadv)
20: ....
21: end subroutine
```

T5: Parallelism for OpenMP (thread manages Accelerators)

T4: Inlining to increase work

T7: Overlap computations & comms.

T5: Data scoping, & outlining final kernel

T3: Prefetch Data to Local Memory or Change Data layout

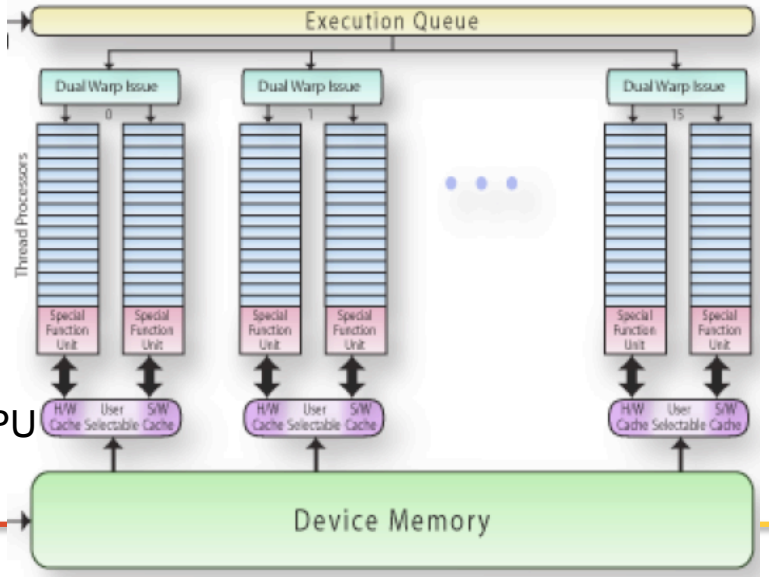
```
1: function divergence_sphere
...
2:     do j=1,nv
3:      do i=1,nv
4:        gv(i,j,1)=elem%metdet(i,j)*(elem%Dinv(1,1,i,j)*..
5:        gv(i,j,2)=elem%metdet(i,j)*(elem%Dinv(2,1,i,j)*..
6:        enddo
7:     enddo
8:     do j=1,nv
9:      do l=1,nv
10:     dudx00=0.0d0
11:     dvdy00=0.0d0
12:       do i=1,nv
13:         dudx00 = dudx00 + deriv%Dvv(i,l  )*gv(i,j  ,1)
14:         dvdy00 = dvdy00 + deriv%Dvv(i,l  )*gv(j  ,i,2)
15:       end do
16:      div(l  ,j  ) = dudx00
17:      vvtemp(j  ,l  ) = dvdy00
18:       end do
19:     end do
20:     do j=1,nv
21:      do i=1,nv
22:         div(i,j)=elem%rmetdetp(i,j)*(rdx*div(i,j) &
23:                  +rdy*vvtemp(i,j))
24:       end do
25:     end do
26:     end function divergence_sphere
```

T2: Loop Fusion to merge kernels, data reuse

T1: Identify Levels of Parallelism for GPU

T0: Port kernel for GPU

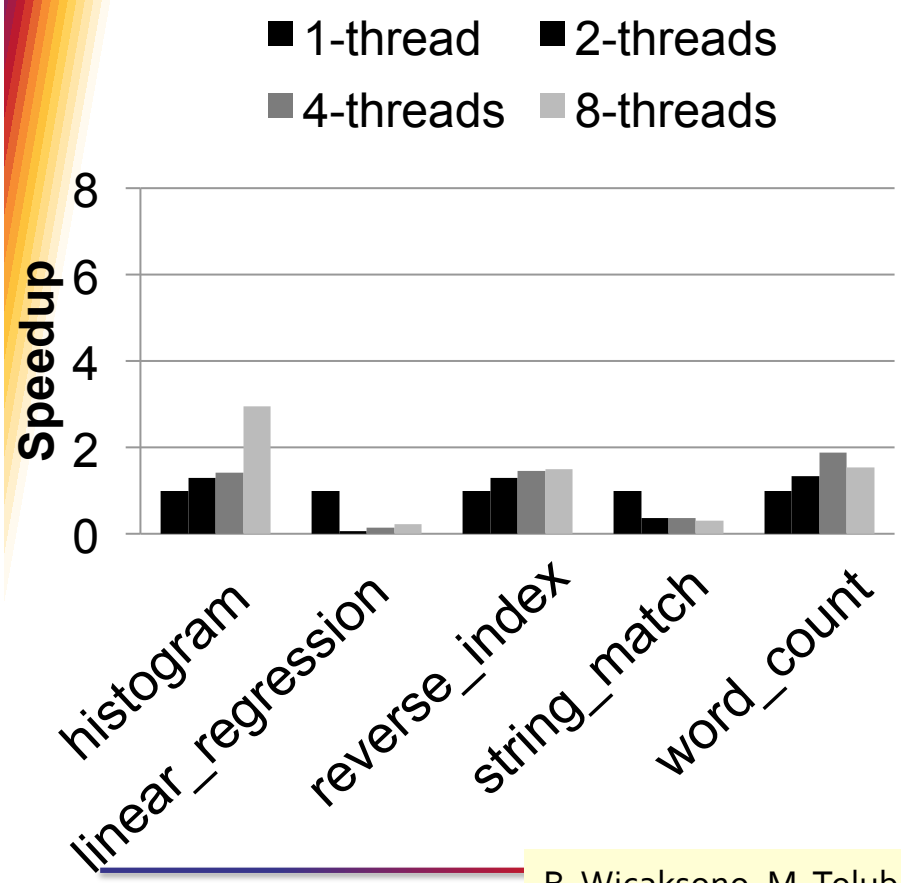
T6: Distribute work across multiple cores/GPUs



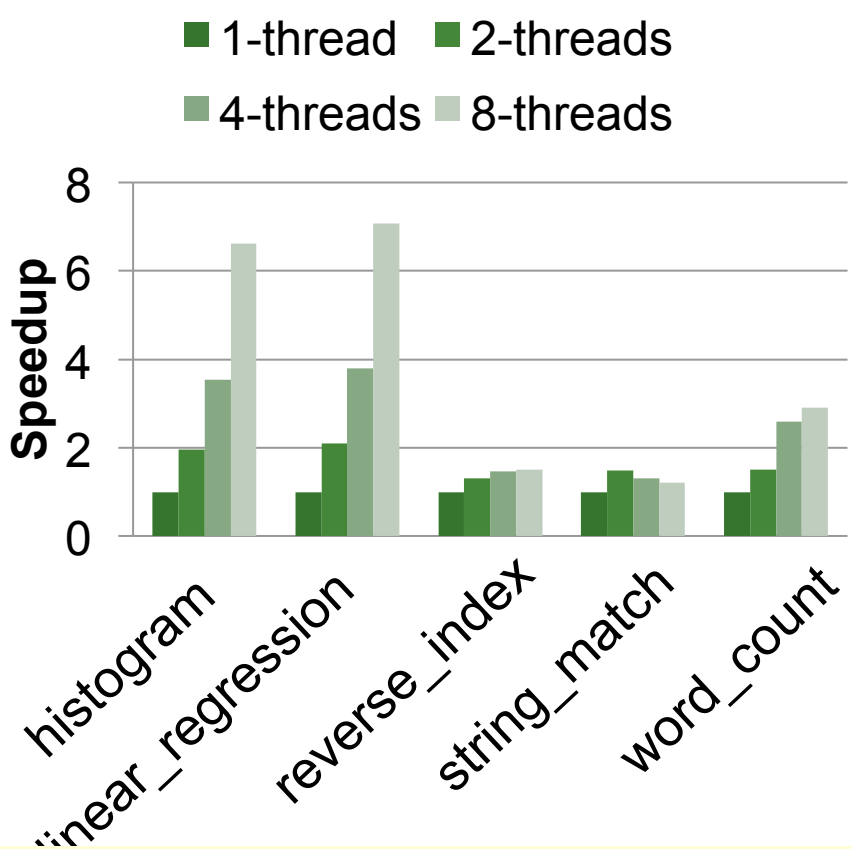

Fermi GPU

# Runtime False Sharing Detection

## Original Version



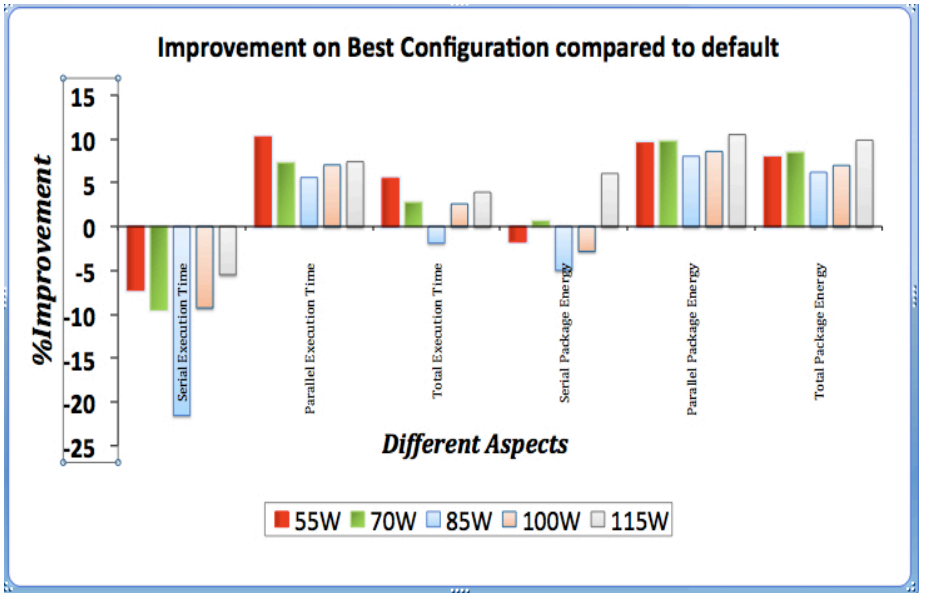

## Optimized Version




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

# Energy Management Tools

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




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

# Where are Directives Headed?

- OpenMP continues to evolve to meet new needs
  - Broad user base; yet strong HPC representation
  - Paying more attention to data locality, access pattern (locality, affinity); it has always mattered for performance
  - A prescriptive model, performance fairly well understood
  - Evolving toward tasking, reducing reliance on barriers
- OpenACC more focussed effort, faster progress
  - Will it be subsumed by OpenMP?
- Might require significant rewriting of code
  - Need tools to help create tasks, obtain high locality, tune for energy and performance

# Questions?