
The Multicore Programming Challenge

Barbara Chapman
University of Houston
November 22, 2007



Agenda

- Multicore is Here ... Here comes Manycore
- The Programming Challenge
- OpenMP as a Potential API
- How about the Implementation?

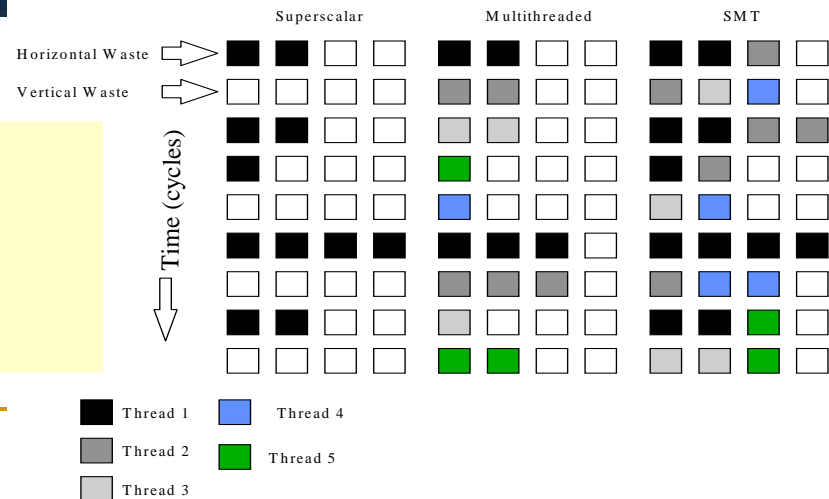


The Power Wall



Want to avoid the heat wave? Go multicore!

Add shared memory multithreading (SMT) for better throughput.
Add accelerators for low power high performance on some operations



The Memory Wall

Growing disparity between memory access times and CPU speed

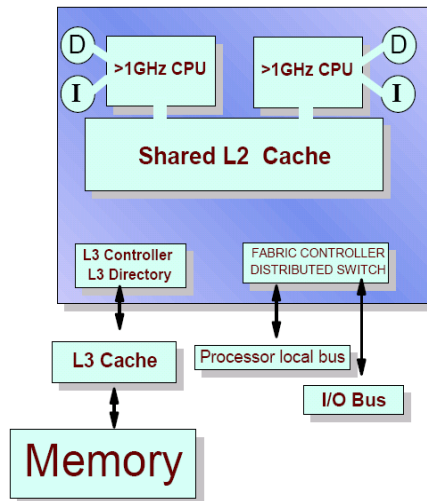
Cache size increases show diminishing returns

Multithreading can help overcome minor delays.

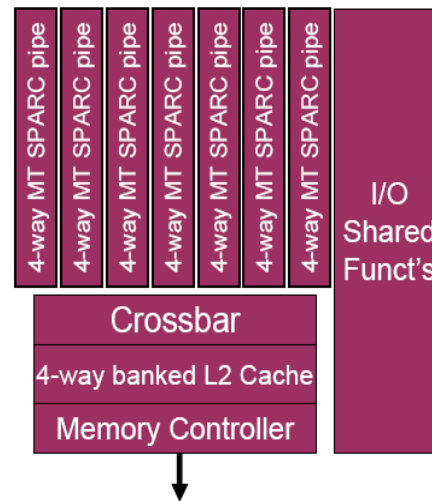
But multicore, SMT reduces amount of cache per thread and introduces competition for bandwidth to memory



So Now We have Multicore



IBM Power4, 2001



Sun T-1 (Niagara), 2005

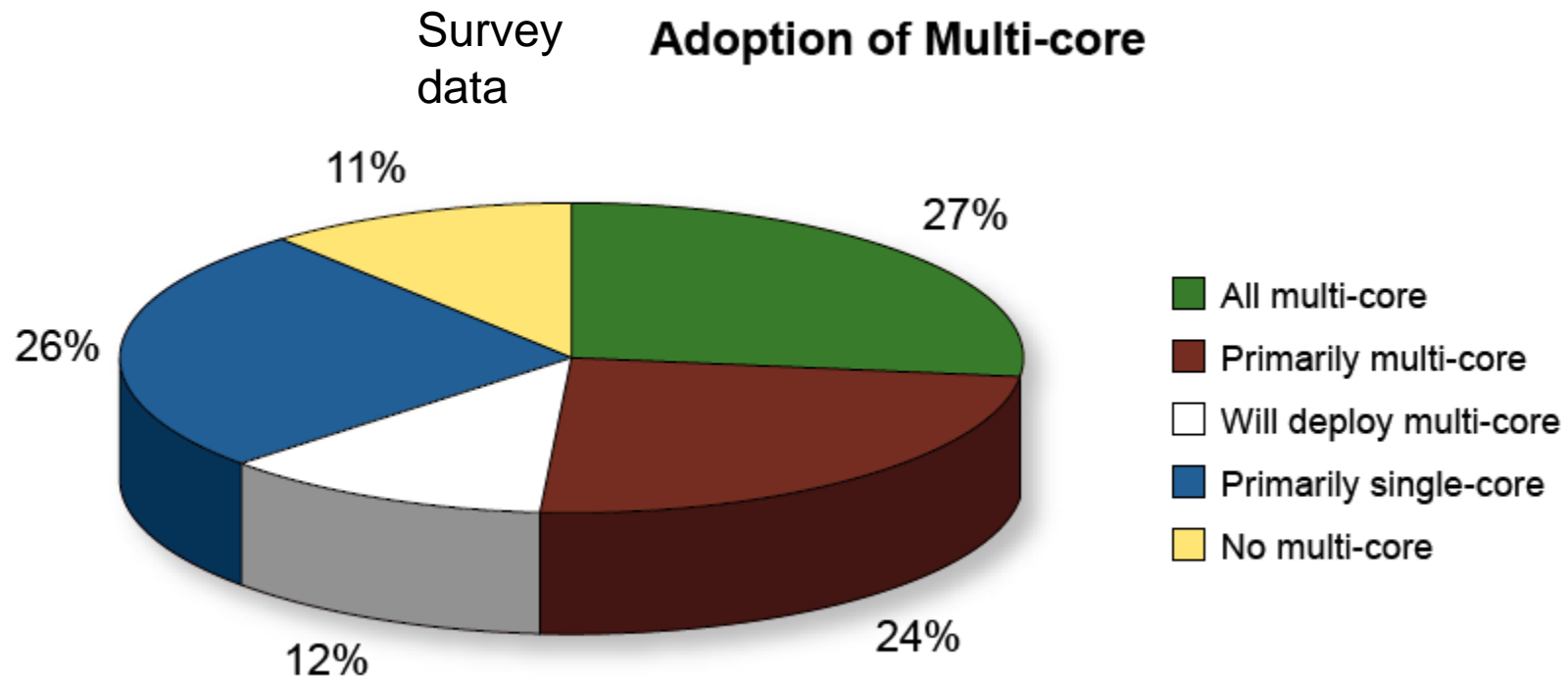


Intel rocks the boat 2005

- Small number of cores, shared memory
- Some systems have multithreaded cores
- Trend to simplicity in cores (e.g. no branch prediction)
- Multiple threads share resources (L2 cache, maybe FP units)
- Deployment in embedded market as well as other sectors

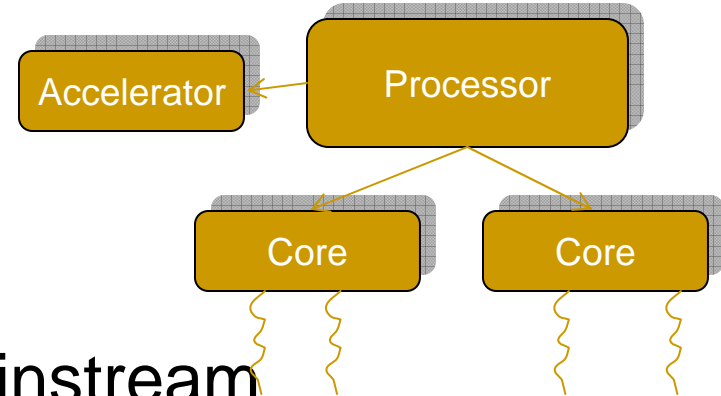
Take-Up in Enterprise Server Market

- Increase in volume of users
- Increase in data, number of transactions
- Need to increase performance



What Is Hard About MC Programming?

We may want sibling threads to share in a workload on a multicore. But we may want SMT threads to do different things



- Parallel programming is mainstream
 - Lower single thread performance
- Hierarchical, heterogeneous parallelism
 - SMPs, multiple cores, SMT, ILP, FPGA, GPGPU,...
 - Diversity in kind and extent of resource sharing, potential for thread contention
 - Reduced effective cache per instruction stream
 - Non-uniform memory access on chip
- Contention for access to main memory
- Runtime power management

Manycore is Coming, Ready or Not

An Intel prediction: technology might support

2010: 16—64 cores 200GF—1 TF

2013: 64—256 cores 500GF— 4 TF

2016: 256--1024 cores 2 TF— 20 TF

- More cores, more multithreading
- More complexity in individual system
 - Hierarchical parallelism (ILP, SMT, core)
 - Accelerators, graphics units, FPGAs
- Multistage networks for data movement and sync.?
- Memory coherence?

Applications are long-lasting: A program written for multicore computers may need to run fast on manycore systems later

Agenda

- Multicore is Here ... Here comes Manycore
- **The Programming Challenge**
- OpenMP as a Potential API
- How about the Implementation?



Application Developer Needs

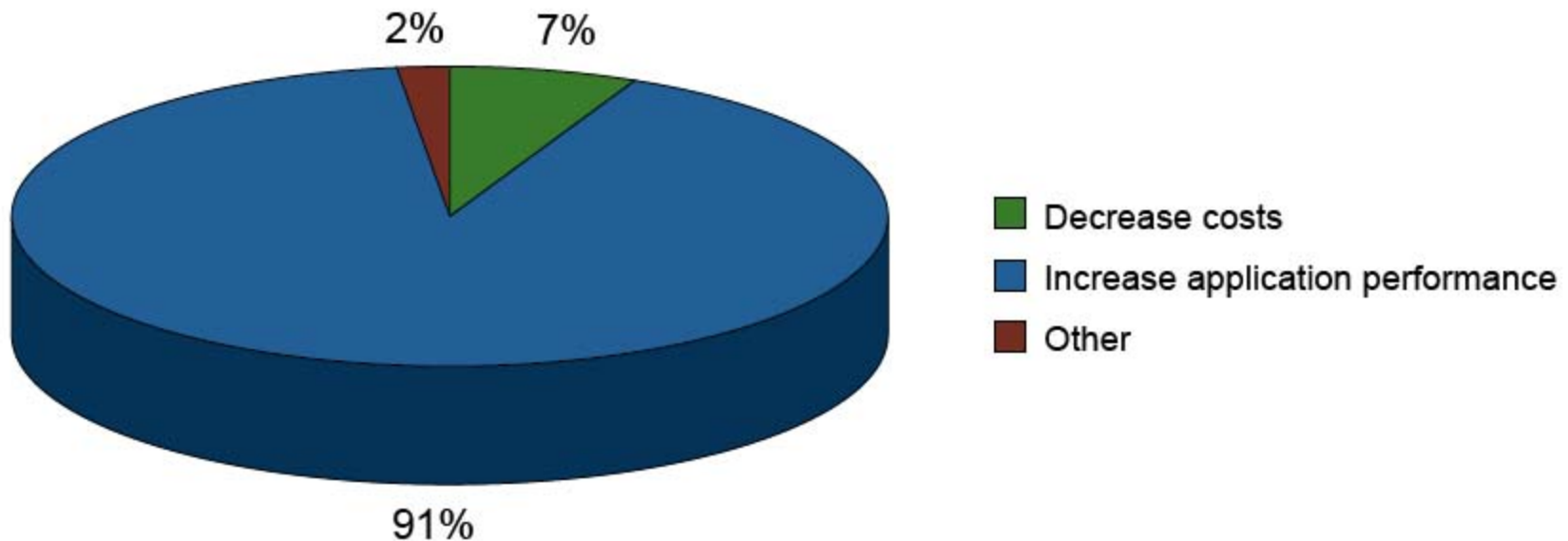
- Time to market
 - Often an overriding requirement for ISVs and enterprise IT
 - May favor rapid prototyping and iterative development
- Cost of software
 - Development , testing
 - Maintenance and support (related to quality but equally to ease of use)
- Human effort of development and testing also now a recognized problem in HPC and scientific computing

Productivity

Does It Matter? Of Course!

Server
market
survey

Reasons for purchasing multi-core hardware



Performance

Programming Model: Some Requirements

- General-purpose platforms are parallel
 - **Generality** of parallel programming model matters
- User expectations
 - **Performance** and **productivity** matter, so does error handling
- Many threads with shared memory
 - **Scalability** matters
- Mapping of work and data to machine will affect performance
 - Work / data **locality** matters
- More complex, “componentized” applications
 - **Modularity** matters

Even if parallelization is easy, scaling might be hard
Amdahl's Law

Some Programming Approaches

- **From high-end computing**
 - **Libraries**
 - MPI, Global Arrays
 - **Partitioned Global Address Space Languages**
 - Co-Array Fortran, Titanium, UPC
- **Shared memory programming**
 - OpenMP, Pthreads, autoparallelization
- **New ideas**
 - **HPCS Languages**
 - Fortress, Chapel, X10
 - **Transactional Memory**

And vendor- and
domain-specific
APIs

Let's explore further...

First Thoughts: Ends of Spectrum

Automatic parallelization

- Usually works for short regions of code
- Current research attempts to do better by combining static and dynamic approaches to uncover parallelism
- Consideration of interplay with ILP-level issues in multithreaded environment

MPI (Message Passing Interface)

- Widely used in HPC, can be implemented on shared memory
 - Enforces locality
 - But lack of incremental development path, relatively low level of abstraction and uses too much memory
 - For very large systems: How many processes can be supported?
-

PGAS Languages

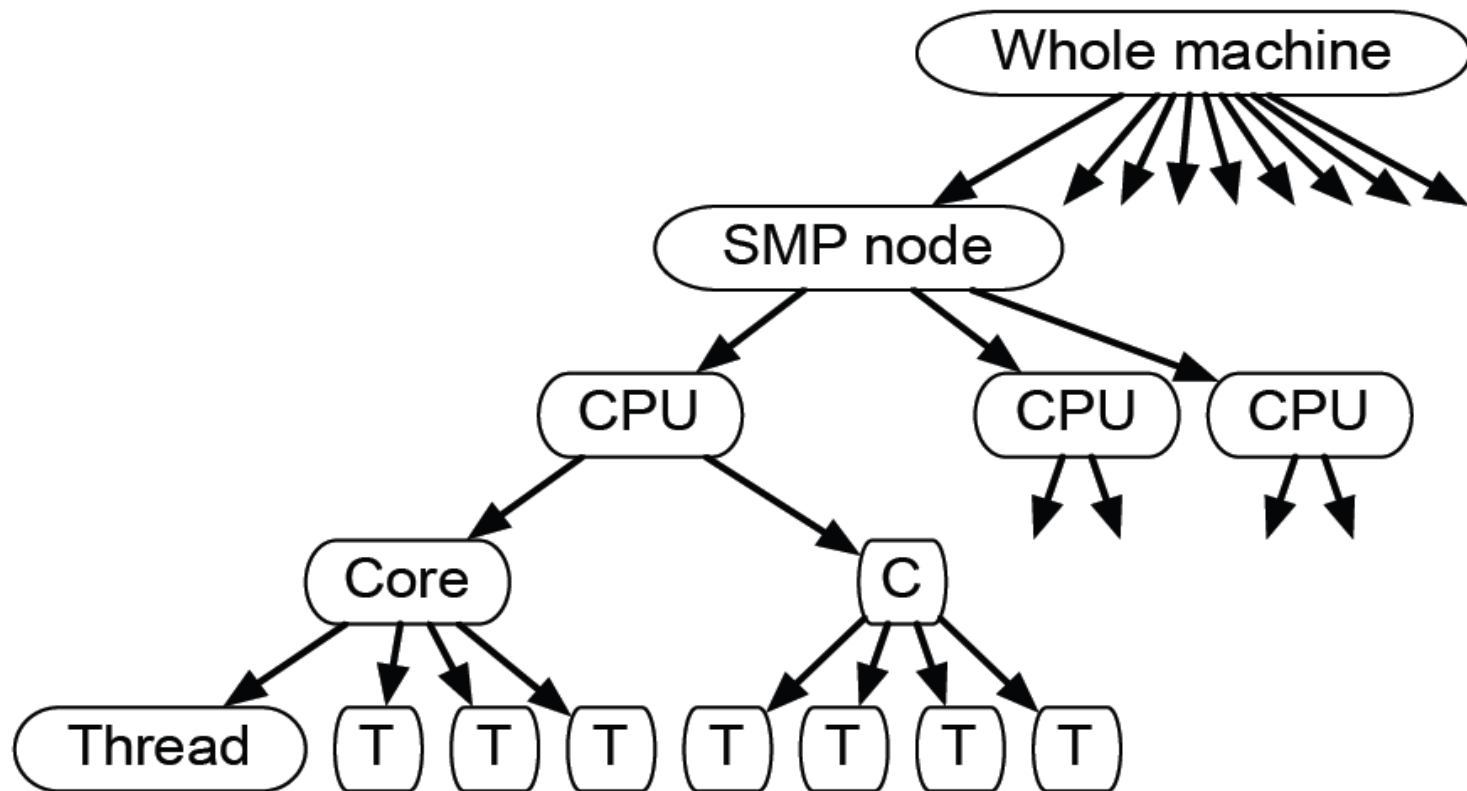
- Partitioned Global Address Space
 - Co-Array Fortran
 - Titanium
 - UPC
- Different details but similar in spirit
 - Raises level of abstraction
 - User specifies data and work mapping
 - Not designed for fine-grained parallelism
- Co-Array Fortran
 - Communication explicit but details up to compiler
 - SPMD computation (local view of code)
 - Entering Fortran standard

$$X = F[p]$$

HPCS Languages

- High-performance High-Productivity Programming
 - **CHAPEL, Fortress, X10**
 - Research languages that explore a variety of new ideas
 - Target global address space, multithreading platforms
 - Aim for high levels of scalability
 - Asynchronous and synchronous threads
 - All of them give priority to support for locality and affinity
 - Machine descriptions, mapping of work and computation to machine
 - Locales, places
 - Attempt to lower cost of synchronization and provide simpler programming model for synchronization
 - Atomic blocks / transactions
-

Machine Abstraction: A Tree





Task Parallelism



- ◆ *co-begins*: indicate statements that may run in parallel:

```
computePivot(lo, hi, data);  
cobegin {  
    Quicksort(lo, pivot, data);  
    Quicksort(pivot, hi, data);  
}  
  
cobegin {  
    ComputeTaskA(...);  
    ComputeTaskB(...);  
}
```

- ◆ *atomic sections*: support atomic transactions

```
atomic {  
    newnode.next = insertpt;  
    newnode.prev = insertpt.prev;  
    insertpt.prev.next = newnode;  
    insertpt.prev = newnode;  
}
```

- ◆ *sync and single-assignment variables*: synchronize tasks
 - similar to Cray MTA C/Fortran



Shared Memory Models 1: PThreads

- Flexible library for shared memory programming
 - Some deficiencies as a result: No memory model
- Widely available
- Does not support productivity
 - Relatively low level of abstraction
 - Doesn't really work with Fortran
 - No easy code migration path from sequential program
 - Lack of structure means error-prone
- Performance can be good

Likely to be used for programming multicore

Agenda

- Multicore is Here ... Here comes Manycore
- The Programming Challenge
- **OpenMP as a Potential API**
- How about the Implementation?



Shared Memory Models 2: OpenMP*

```
C$OMP FLUSH
```

```
#pragma omp critical
```

```
C$OMP THREADPRIVATE(/ABC/)
```

```
CALL OMP SET NUM THREADS(10)
```

- A set of compiler directives and library routines
 - Can be used with Fortran, C and C++
- User maps code to threads that share memory
 - Parallel loops, parallel sections, workshare
- User decides if data is shared or private
- User coordinates shared data accesses
 - Critical regions, atomic updates, barriers, locks

```
C$OMP
```

```
C$OMP
```

```
C$OMP
```

```
D
```

```
C
```

```
#p
```

```
OMP BARRIER
```

```
C$OMP PARALLEL COPYIN(/blk/)
```

```
C$OMP DO lastprivate(XX)
```

```
Nthrds = OMP_GET_NUM_PROCS()
```

```
omp_set_lock(lck)
```

* The name "OpenMP" is the property of the OpenMP Architecture Review Board.

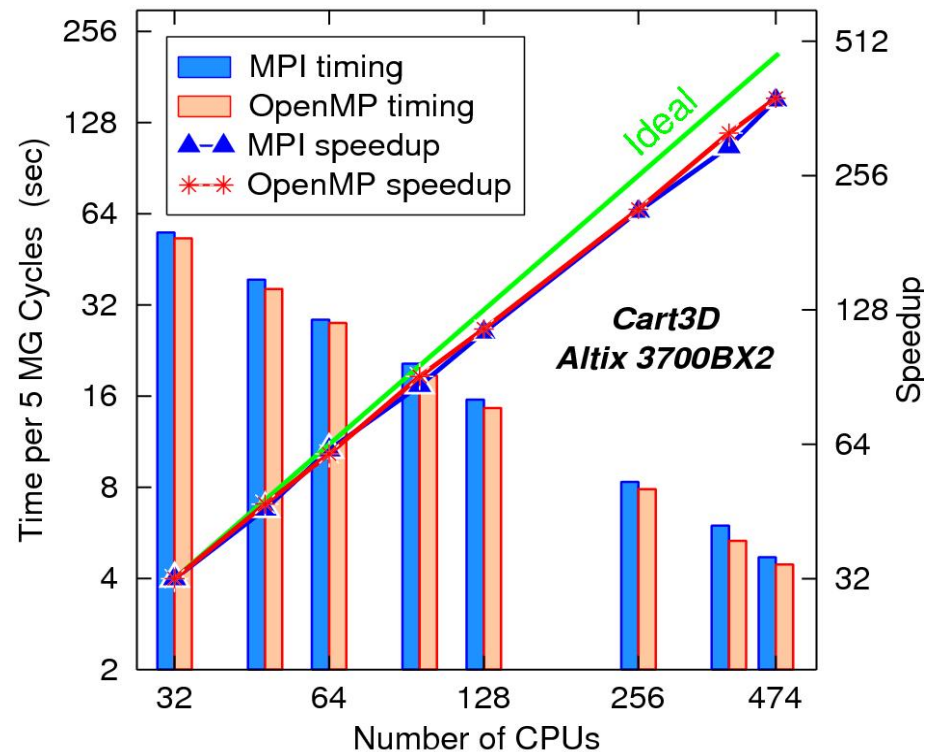
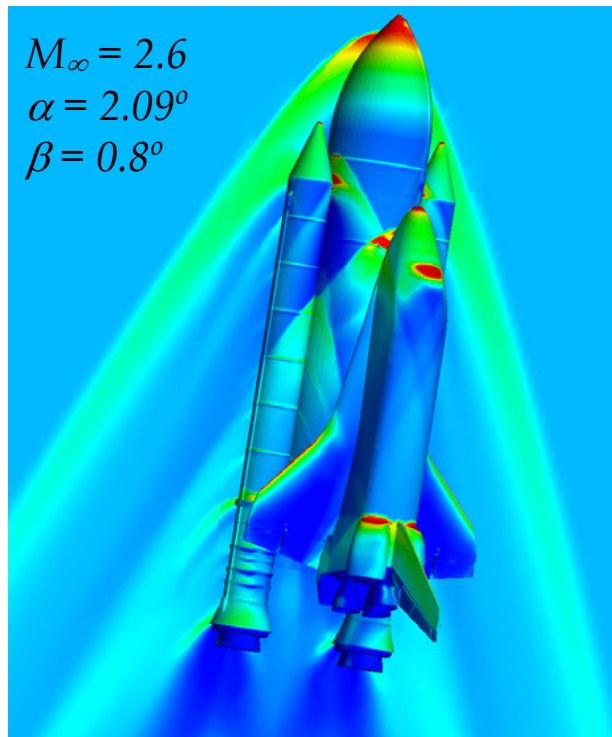
Shared Memory Models 2: OpenMP

- High-level directive-based multithreaded programming
 - The user makes strategic decisions
 - Compiler figures out details
 - Threads interact by sharing variables
 - Synchronization to order accesses and prevent data conflicts
 - Structured programming to reduce likelihood of bugs

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

Cart3D OpenMP Scaling

4.7 M cell mesh Space Shuttle Launch Vehicle example



- OpenMP version uses same domain decomposition strategy as MPI for data locality, avoiding false sharing and fine-grained remote data access
- OpenMP version slightly outperforms MPI version on SGI Altix 3700BX2, both close to linear scaling.

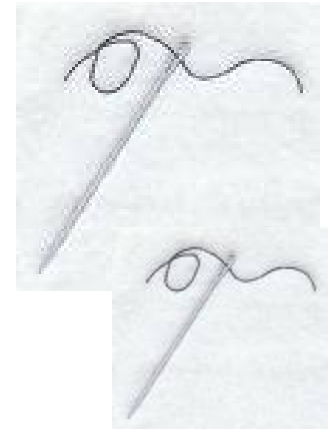


The OpenMP ARB

- 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, Cray, Fujitsu, HP, IBM, Intel, Microsoft, NEC, PGI, SGI, Sun
 - Auxiliary: ASCI, cOMPunity, EPCC, KSL, NASA, RWTH Aachen

OpenMP

- Oct 1997 – 1.0 Fortran
 - Oct 1998 – 1.0 C/C++
 - Nov 1999 – 1.1 Fortran (interpretations added)
 - Nov 2000 – 2.0 Fortran
 - Mar 2002 – 2.0 C/C++
 - May 2005 – 2.5 Fortran/C/C++ (mostly a merge)
 - ?? 2008 – 3.0 Fortran/C/C++ (extensions)
-
- Original goals:
 - Ease of use, incremental approach to parallelization
 - “Adequate” speedup on small SMPs, ability to write scalable code on large SMPs with corresponding effort
 - As far as possible, parallel code “compatible” with serial code





OpenMP 3.0

- Many proposals for new features
- Features to enhance expressivity, expose parallelism, support multicore
 - Better parallelization of loop nests
 - Parallelization of wider range of loops
 - Nested parallelism
 - Controlling the default behavior of idle threads
- And more

Pointer -chasing Loops in OpenMP?

```
for(p = list; p; p = p->next) {  
    process(p->item);  
}
```

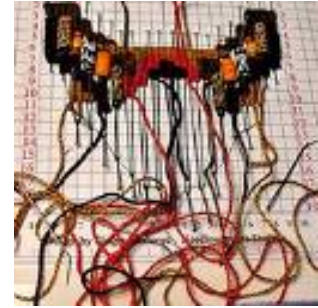
- Cannot be parallelized with `omp for`: number of iterations not known in advance
 - Transformation to a “canonical” loop can be very labor-intensive/inefficient
-

OpenMP 3.0 Introduces Tasks

- Tasks explicitly created and processed
 - Each encountering thread packages a new instance of a task (code and data)
 - Some thread in the team executes the task

```
#pragma omp parallel
{
  #pragma omp single
  {
    p = listhead ;
    while (p) {
      #pragma omp task
        process (p)
      p=next (p) ;
    }
  }
}
```

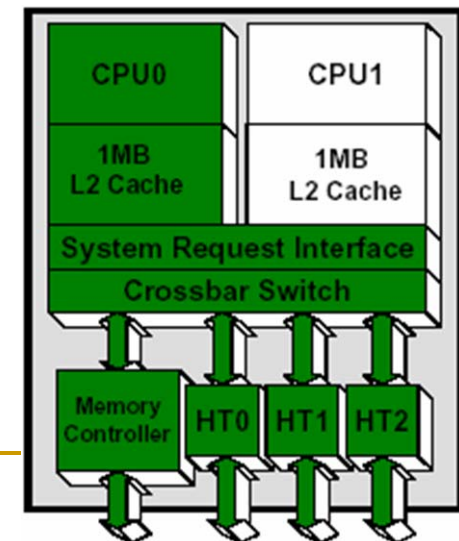
More and More Threads



- Busy waiting may consume valuable resources, interfere with work of other threads on multicore
 - OpenMP 3.0 will allow user more control over the way idle threads are handled
 - Improved support for multilevel parallelism
 - More features for nested parallelism
 - Different regions may have different defaults
 - E.g. `omp_set_num_threads()` inside a parallel region.
 - Library routines to determine depth of nesting and IDs of parent/grandparent threads.
-

What is Missing? 1: 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 wherever possible, and optimizing code for cache
 - This can work pretty well
 - But small mistakes may be costly

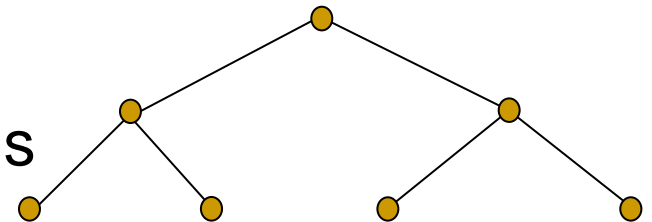


Relies on OS for mapping threads to system

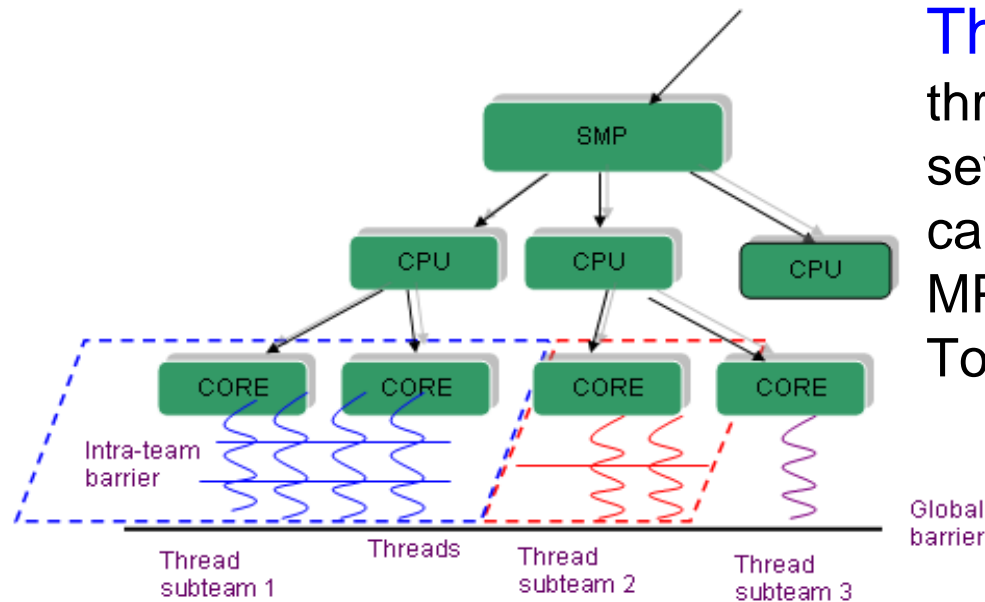
Locality Support: An Easy Way



- A simple idea for data placement
 - Rely on implicit first touch or other system support
 - Possibly optimize e.g. via preprocessing step
 - Provide a “next touch” directive that would store data so that it is local to next thread accessing it
- Allow programmer to give hints on thread placement
 - “spread out”, “keep close together
 - Logical machine description?
 - Logical thread structure?
- Nested parallelism causes problems
 - Need to place all threads
 - But when mapping initial threads, those at deeper levels have not been created



OpenMP Locality: Thread Subteams

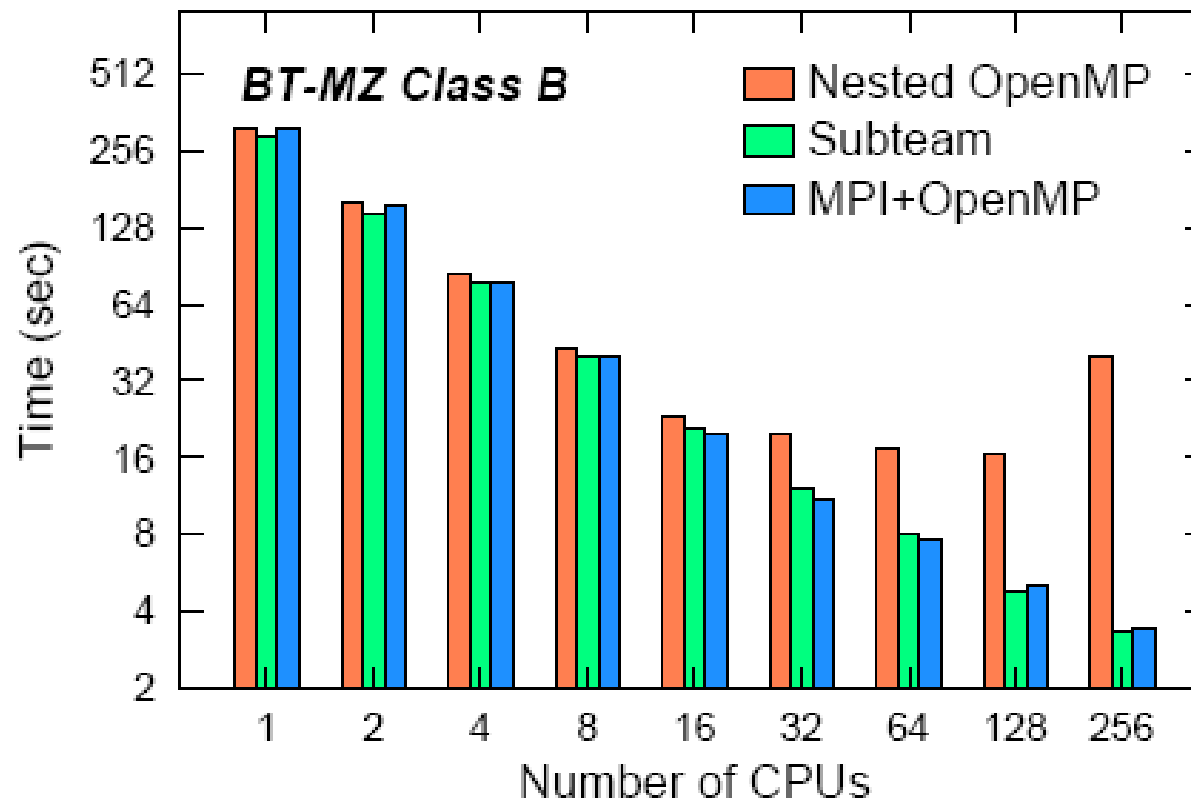


Thread Subteam: original thread team is divided into several subteams, each of which can work simultaneously. Like MPI process groups. Topologies could also be defined.

- Flexible parallel region/worksharing/synchronization extension
- Low overhead because of static partition
- Facilitates thread-core mapping for better data locality and less resource contention
- Supports heterogeneity, hybrid programming, composition

#pragma omp for on threads (m:n:k)

BT-MZ Performance with Subteams



Platform: Columbia@NASA



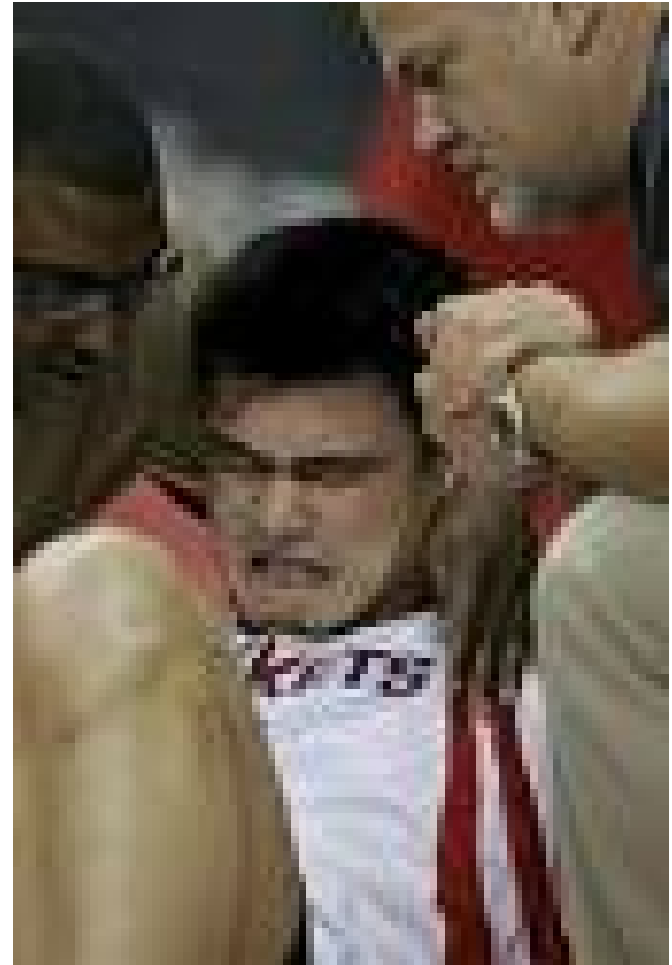
Subteam: subset of existing team

What is Missing? 2: Synchronization

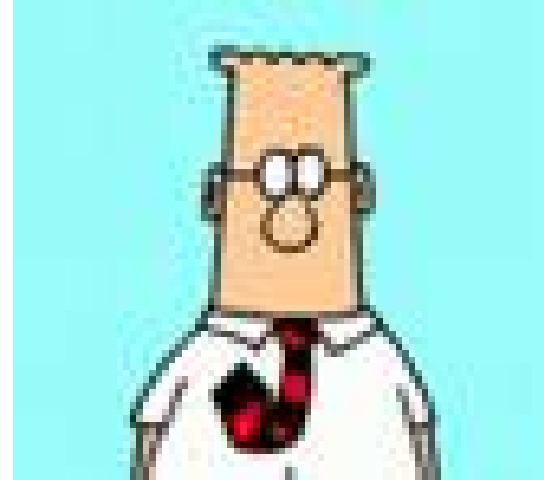
- Reliance on global barriers, critical regions and locks
- Critical region is very expensive
 - High overheads to protect often just a few memory accesses
- It's hard to get locks right
 - And they may also incur performance problems
- Transactions / atomic blocks might be an interesting addition
 - For some applications
 - Especially if hardware support provided

Agenda

- Multicore is Here ... Here comes Manycore
- The Programming Challenge
- OpenMP as a Potential API
- How about the Implementation?



Compiling for Multicore

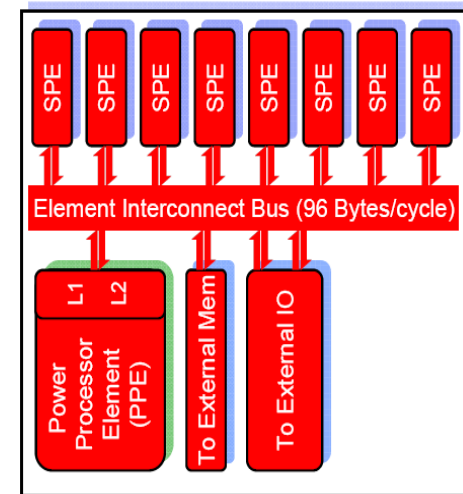


- OpenMP is easy to implement
 - On SMPs
- Runtime system is important part of implementation
- Implementations may need to re-evaluate strategies for multicore
 - Default scheduling policies
 - Choice of number of threads for given computation
- There are some general challenges for compilers on multicore
 - How does it impact instruction scheduling, loop optimization?

Implementation Not Easy Everywhere

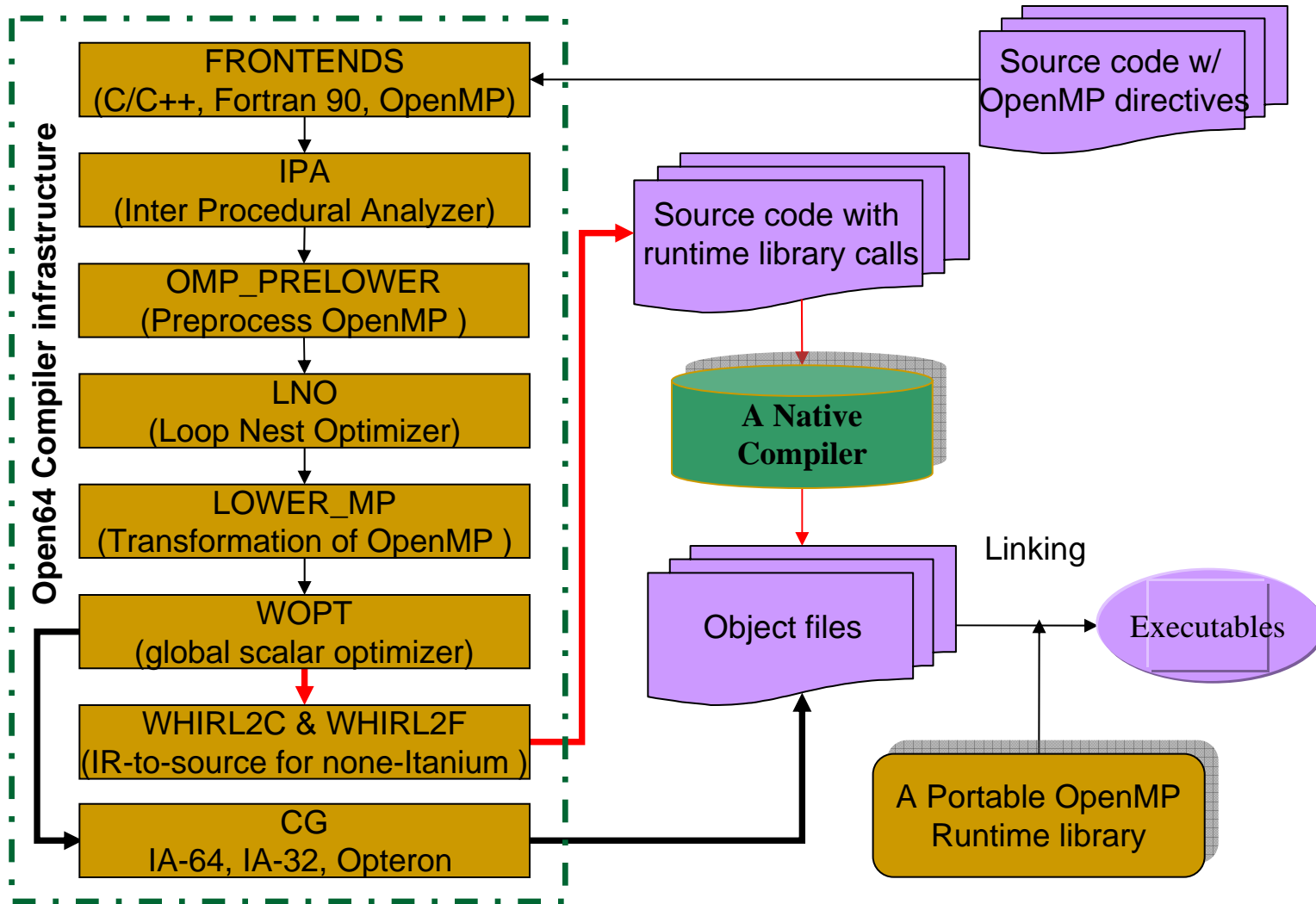
- ❑ Can we provide one model across the board?
- ❑ Indications that application developers would like this

- Some problems:
 - Memory transfers to/from accelerator
 - Size of code
 - Single vs double precision
- When is it worth it?
- Can the compiler overlap copy-ins and computation?
- Do we need language extensions?



Very little memory on SPEs of Cell, cores of ClearSpeed CSX600

OpenUH Compiler Infrastructure



OpenMP Compiler Optimizations

- Most compilers perform optimizations after OpenMP constructs have been lowered
 - Limits traditional optimizations
 - Misses opportunities for high level optimizations

```
#pragma omp parallel
{
  #pragma omp single
  {
    k = 1 ;
  }
  if ( k==1) . . .
}
```

K==1?
Yes

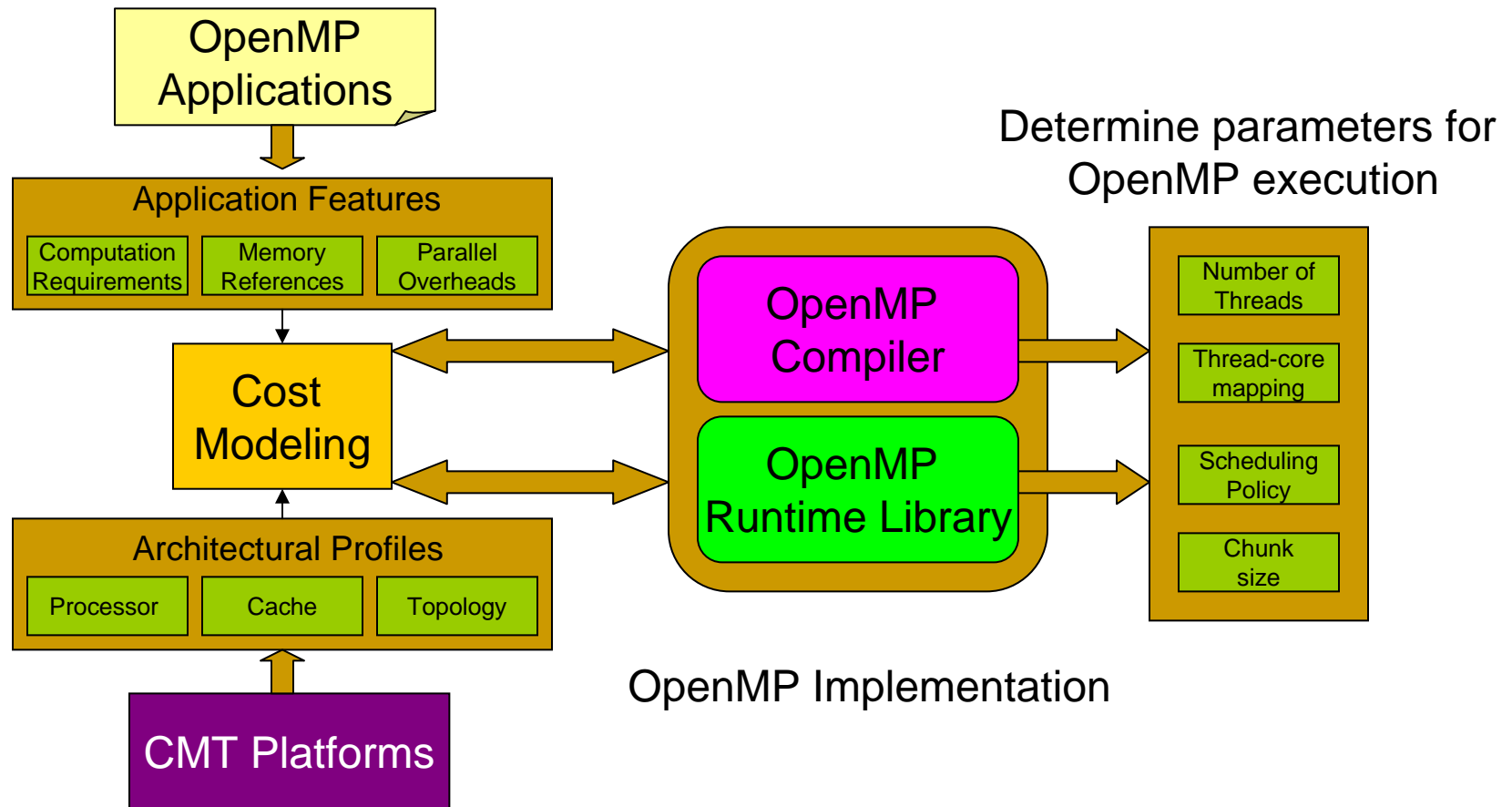
(a) An OpenMP program with a single construct

```
mbsp_status =
ompc_single(ompv_temp_gtid) ;
if (mbsp_status == 1)
{
  k = 1 ;
}
ompc_end_single ( ompv_temp_gtid ,
if ( k==1) . . .
```

K==1?
Unkown

(b) The corresponding compiler translated threaded code

Multicore OpenMP Cost Modeling

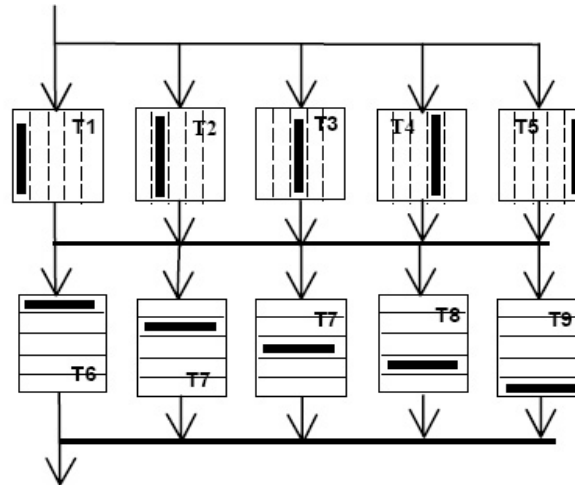


OpenMP Macro-Pipelined Execution

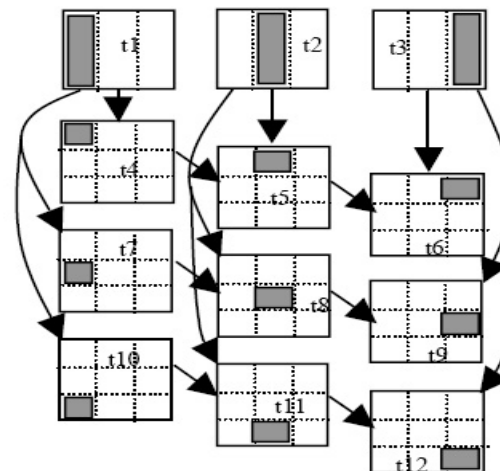
```

!$OMP PARALLEL
!$OMP DO
  do j= 1, N
    do i= 2, N
      A(i,j)=A(i,j)-B(i,j)*A(i-1,j)
    end do
  end do
!$OMP END DO
!$OMP DO
  do i= 1, N
    do j= 2, N
      A(i,j)=A(i,j)-B(i,j) * A(i,j-1)
    end do
  end do
!$OMP END DO
    
```

(a) ADI OpenMP Kernel



(b) Standard OpenMP Execution



(c) Macro-pipelined execution

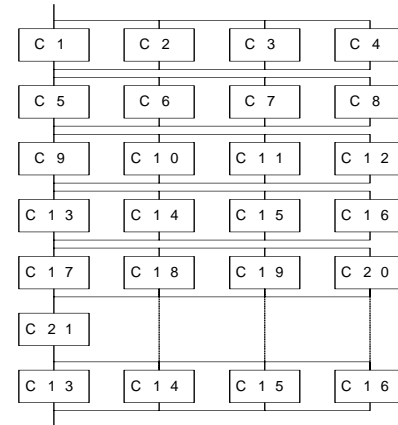
A Longer Term View: Task Graph

```

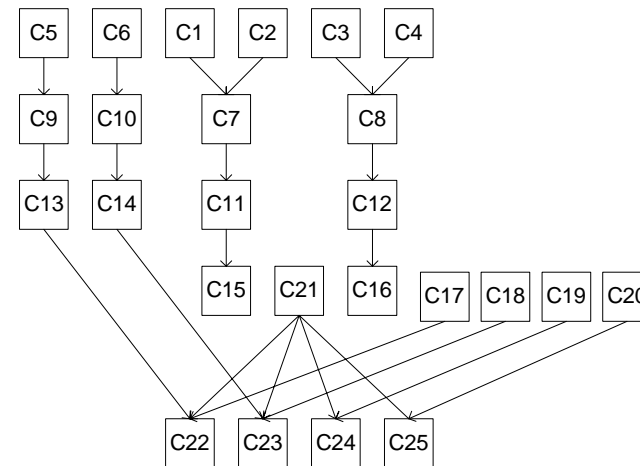
!$OMP PARALLEL
!$OMP DO
  do i=1,imt
    RHOKX(imt,i) = 0.0
  enddo
!$OMP ENDDO
!$OMP DO
  do i=1, imt
    do j=1, jmt
      if (k .le. KMU(j,i)) then
        RHOKX(j,i) = DXUR(j,i)*p5*RHOKX(j,i)
      endif
    enddo
  enddo
!$OMP ENDDO
!$OMP DO
  do i=1, imt
    do j=1, jmt
      if (k > KMU(j,i)) then
        RHOKX(j,i) = 0.0
      endif
    enddo
  enddo
!$OMP ENDDO
  if (k == 1) then
!$OMP DO
  do i=1, imt
    do j=1, jmt
      RHOKMX(j,i) = RHOKX(j,i)
    enddo
  enddo
!$OMP ENDDO
!$OMP DO
  do i=1, imt
    do j=1, jmt
      SUMX(j,i) = 0.0
    enddo
  enddo
!$OMP ENDDO
  endif
!$OMP SINGLE
  factor = dzw(kth-1)*grav*p5
!$OMP END SINGLE
!$OMP DO
  do i=1, imt
    do j=1, jmt
      SUMX(j,i) = SUMX(j,i) + factor * &
        (RHOKX(j,i) + RHOKMX(j,i))
    enddo
  enddo

```

Part of computation of gradient of hydrostatic pressure in POP code

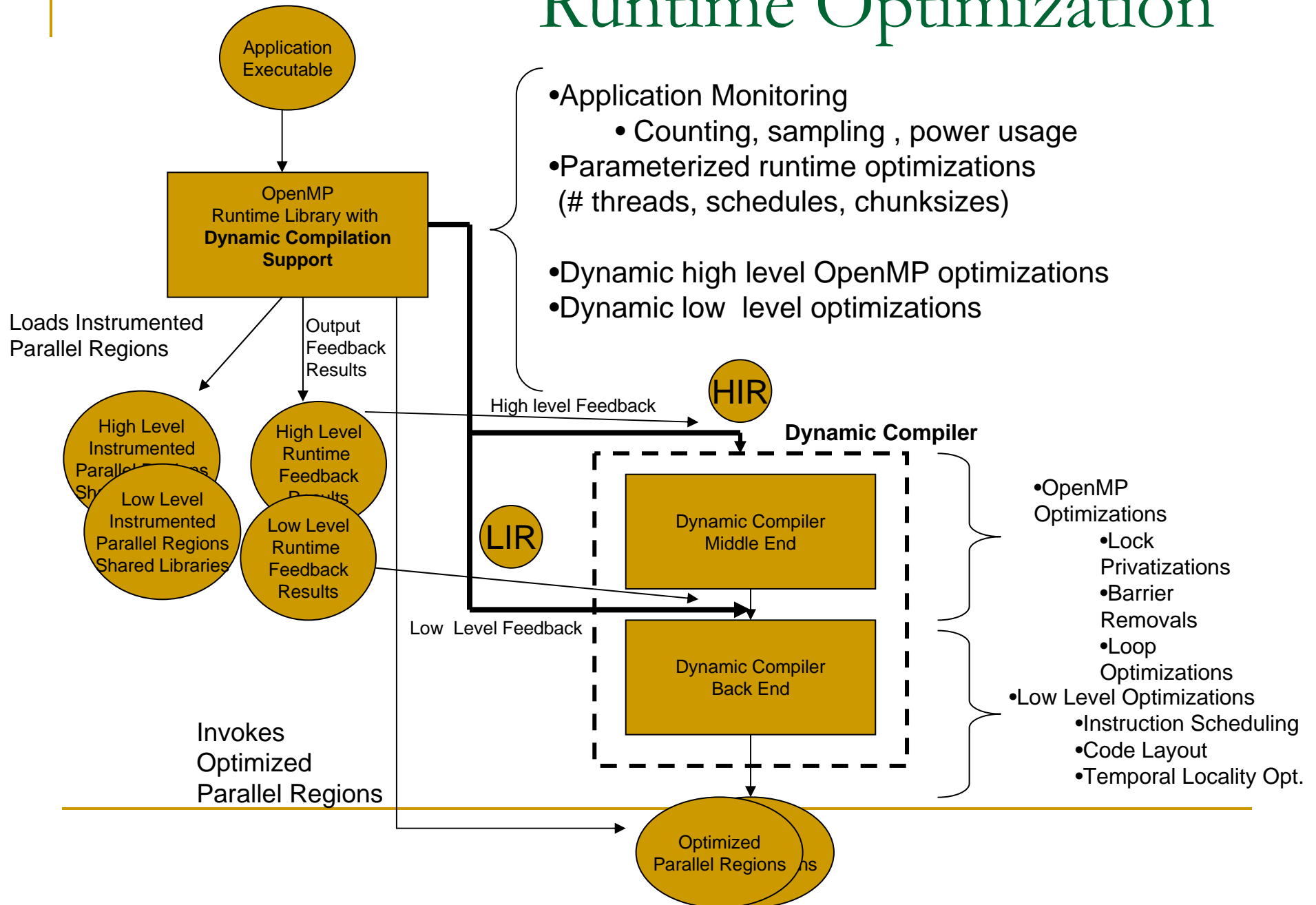


Runtime execution model for (c stands for chunk)



Dataflow execution model associated with translated code

Runtime Optimization



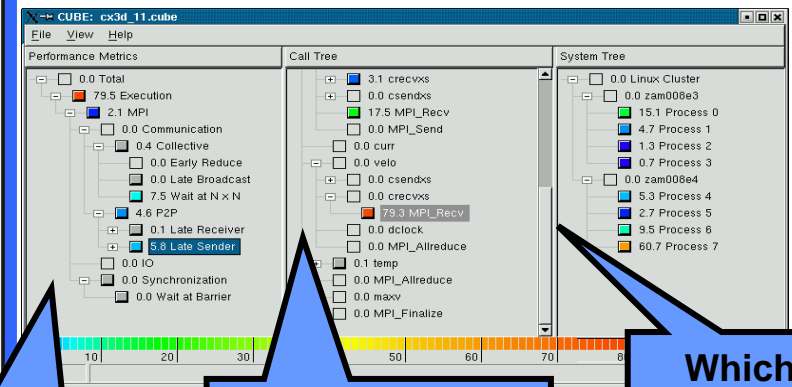
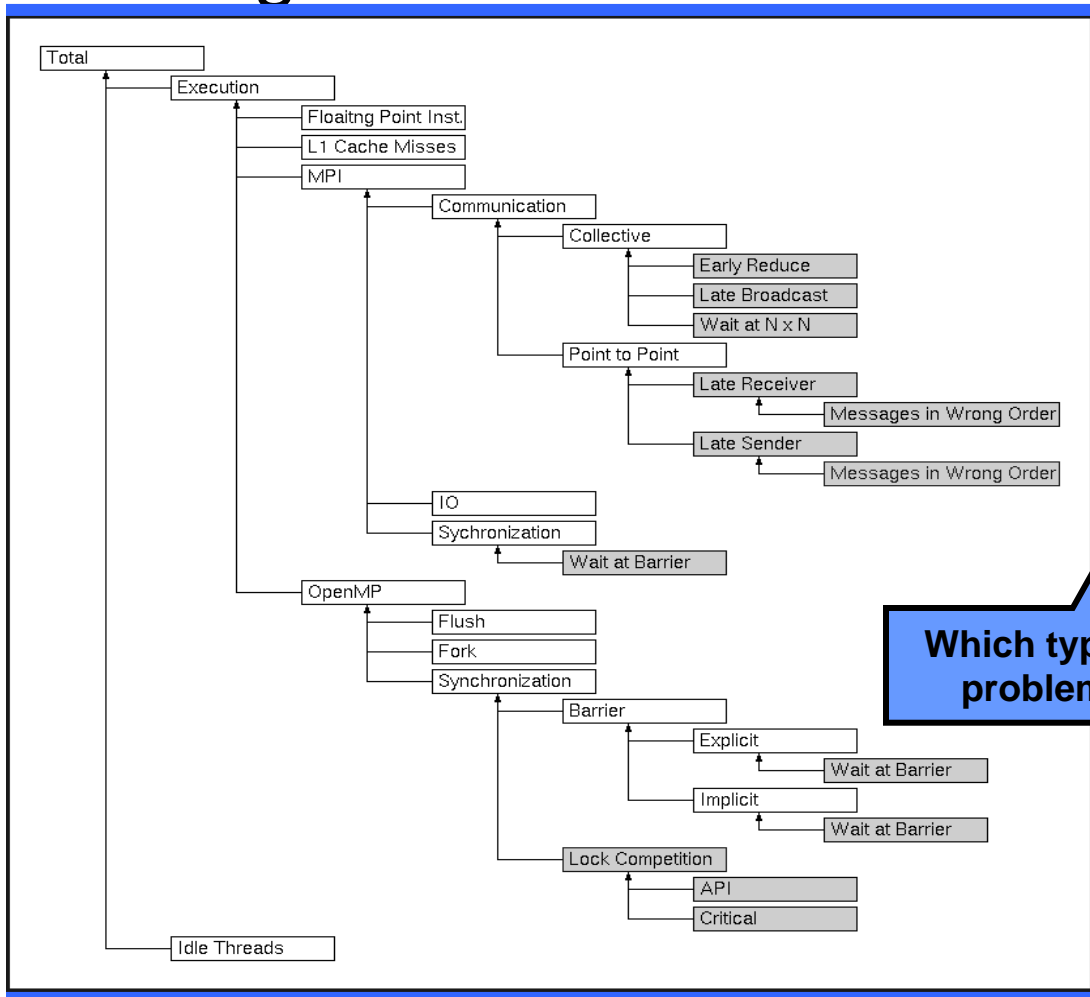
What About The Tools?

- Programming APIs need tool support
 - At appropriate level of abstraction
- OpenMP needs (especially):
 - Support for creation of OpenMP code with high level of locality
 - Data race detection (prevent bugs)
 - Performance tuning at high level especially with respect to memory usage



Analysis via Tracing (KOJAK)

High Level Patterns for MPI and OpenMP

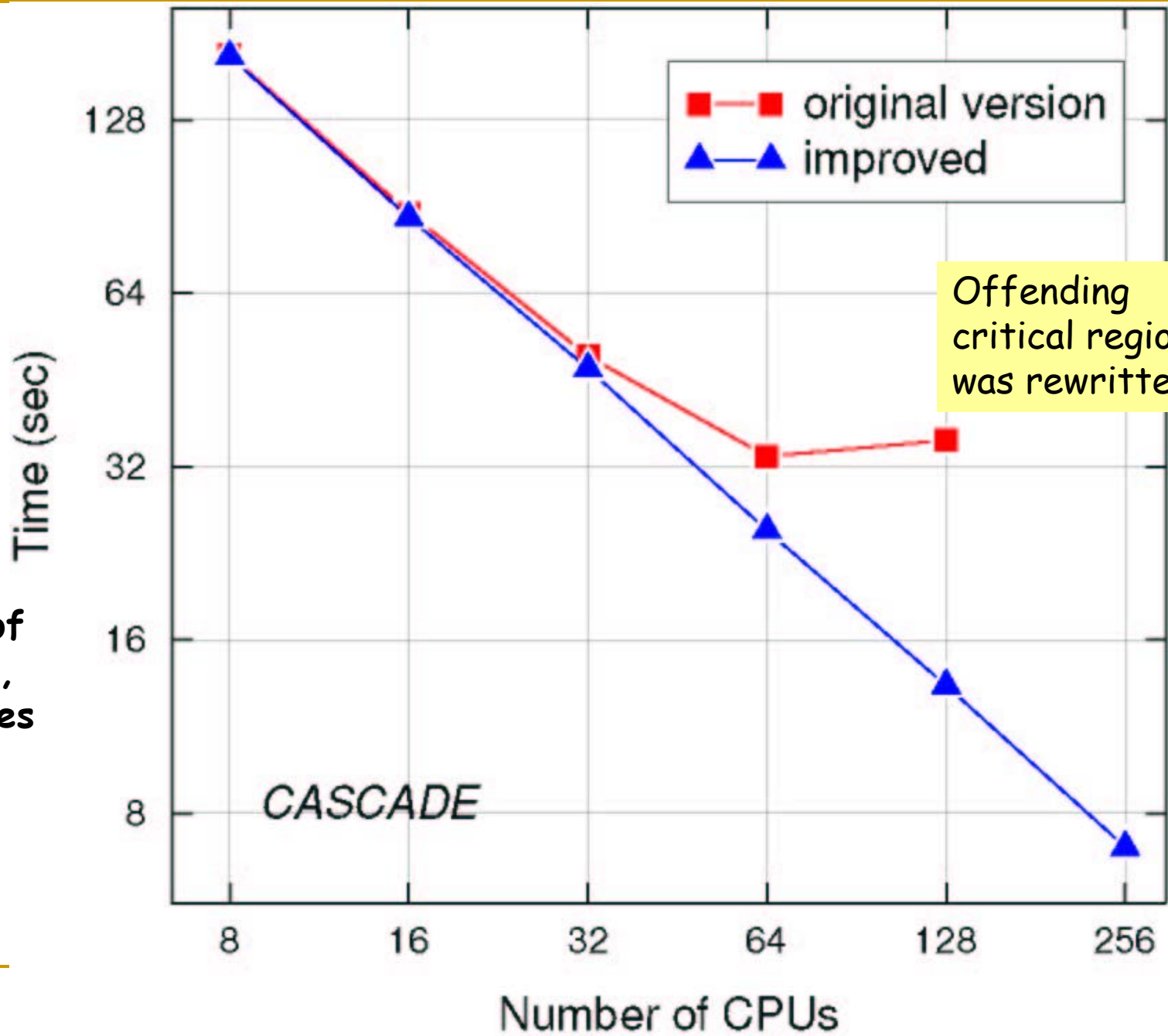


Which type of problem?

Where in the source code? Which call path?

Which process / thread ?

Call tree



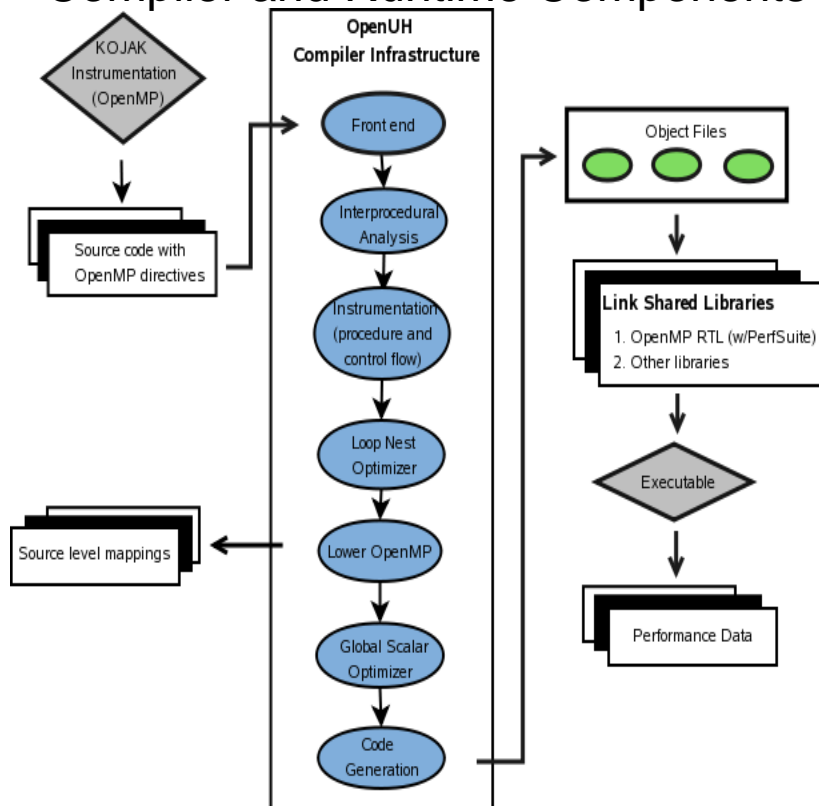
Courtesy of
R. Morgan,
NASA Ames



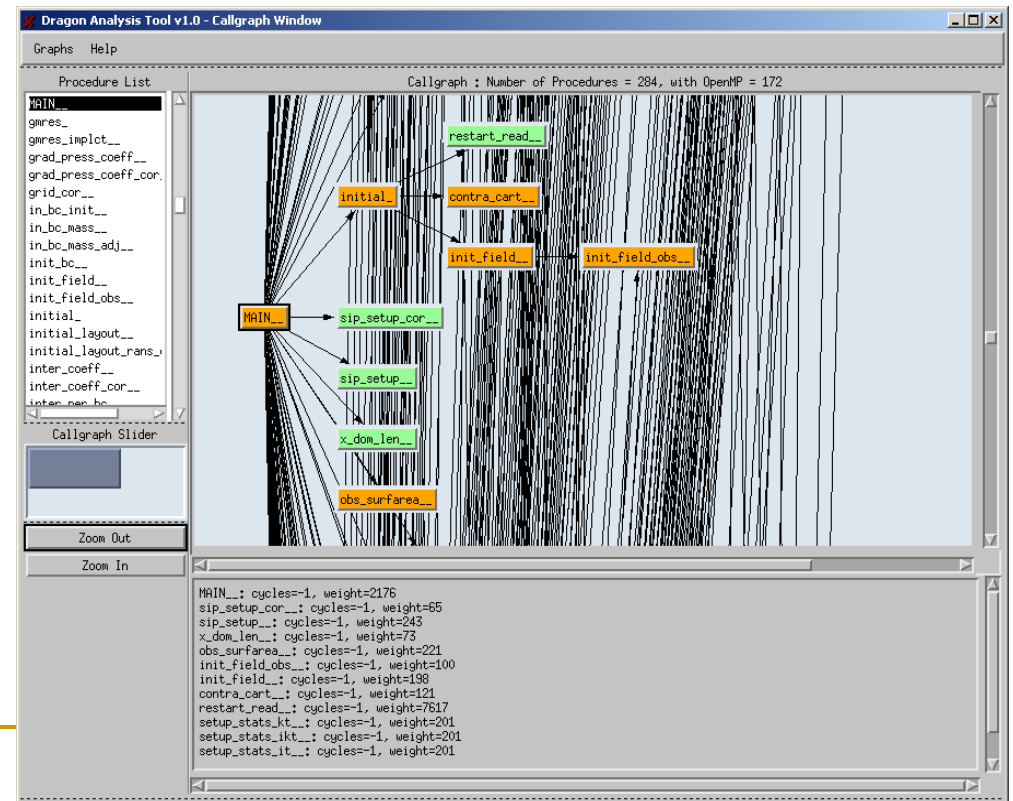
OpenUH Tuning Environment

- Manual or automatic selective instrumentation, possibly in iterative process
- Instrumented OpenMP runtime can monitor parallel regions at low cost
- KOJAK able to look for performance problems in output and present to user

Compiler and Runtime Components



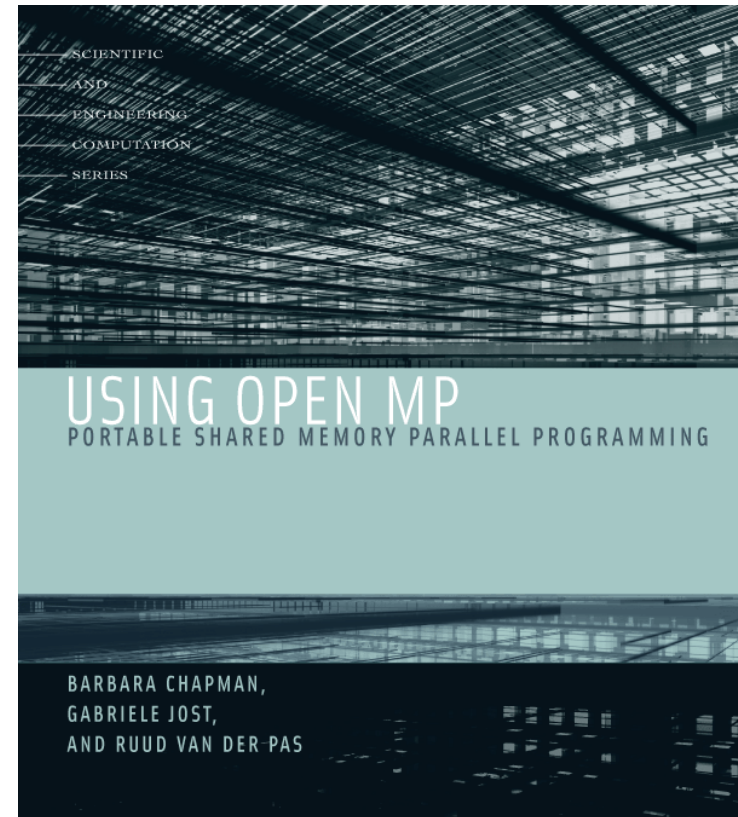
Selective Instrumentation analysis



Don't Forget Training

Teach programmers

- about parallelism
- how to get performance
- Set appropriate expectations



<http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11387>

Summary

- New generation of hardware represents major change
 - Ultimately, applications will have to adapt
 - Application developer should be able to rely on appropriate programming languages, compilers, libraries and tools
 - Plenty of work is needed if these are to be delivered
 - OpenMP is a promising high-level programming model for multicore and beyond
-

Questions?

