Lessons from the past, challenges ahead, and a path forward

John Mellor-Crummey

Department of Computer Science
Rice University

On Programming Models for the Exascale ...

- **Problem: rise of complexity of exascale systems**
- **Idea: provide a high level of abstraction**
  - **—handle mapping onto heterogeneous nodes**
    - **fat multicore + thin manycore**
  - **—handle details of data movement and synchronization**
  - **—handle details of computation partitioning**

A Cautionary Tale ...

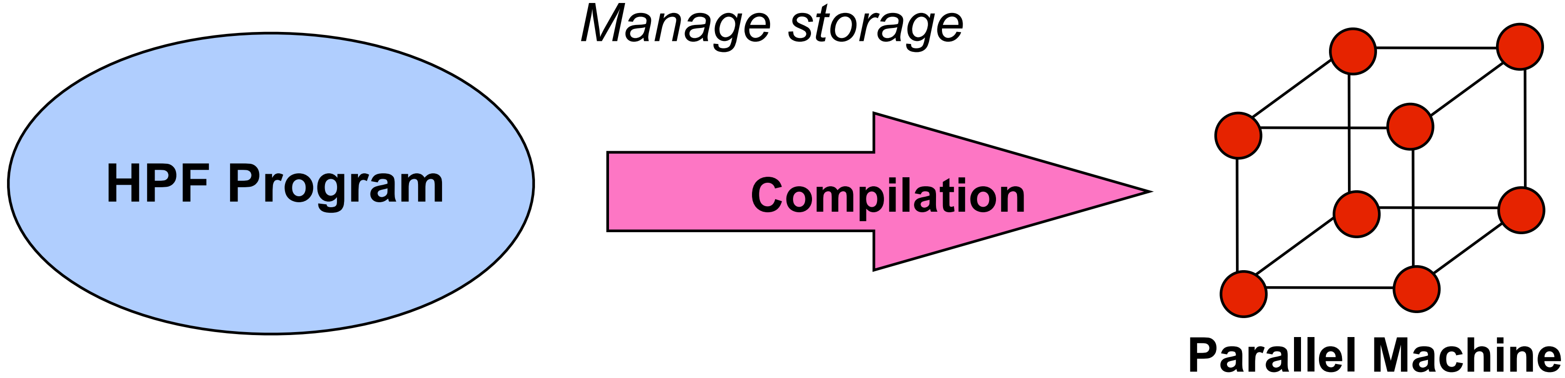
A Decade Ago: High Performance Fortran

Partitioning of data drives partitioning of computation, communication, and synchronization

Fortran program
+ data partitioning

Partition computation
Insert communication
Manage storage

Same answers as
sequential program



Rice dHPF Compiler, circa 2000

- **Sophisticated data partitionings**
  - skewed cyclic tilings using symbolically-parameterized tiles of uneven size with many-one mappings of tiles to processors
- **Sophisticated computation partitionings**
  - e.g. partially-replicated computation to reduce communication
- **Program analysis**
  - polyhedral analysis of iteration spaces, communication
- **Communication optimization**
  - communication normalization, coalescing
  - latency hiding
- **Node performance**
  - generate clean inner loops
  - cache optimization (padding, communication buffer mgmt)

Productive Parallel 1D FFT ($n = 2^k$)

```
subroutine fft(c, n)
    implicit complex(c)
    dimension c(0:n-1), irev(0:n-1)
!HPF$ processors p(number_of_processors())
!HPF$ template t(0:n-1)
!HPF$ align c(i) with t(i)
!HPF$ align irev(i) with t(i)
!HPF$ distribute t(block) onto p
    two_pi = 2.0d0 * acos(-1.0d0)
    levels = number_of_bits(n) - 1
    irev = (/ (bitreverse(i,levels), i= 0, n-1) /)
    forall (i=0:n-1) c(i) = c(irev(i))
    do l = 1, levels                        ! --- for each level in the FFT
       m = ishft(1, l)
       m2 = ishft(1, l - 1)
       do k = 0, n - 1, m                   ! --- for each butterfly in a level
          do j = k, k + m2 - 1              ! --- for each point in a half bfly
             ce = exp(cmplx(0.0,(j - k) * -two_pi/real(m)))
             cr = ce * c(j + m2)
             cl = c(j)
             c(j) = cl + cr
             c(j + m2) = cl - cr
          end do
       end do
    enddo
 end subroutine fft
```

Productive Parallel 1D FFT (n = 2^k)

```
subroutine fft(c, n)
    implicit complex(c)
    dimension c(0:n-1), irev(0:n-1)
!HPF$ processors p(number_of_processors())
!HPF$ template t(0:n-1)
!HPF$ align c(i) with t(i)
!HPF$ align irev(i) with t(i)
!HPF$ distribute t(block) onto p
    two_pi = 2.0d0 * acos(-1.0d0)
    levels = number_of_bits(n) - 1
    irev = (/ (bitreverse(i,levels), i= 0, n-1) /)
    forall (i=0:n-1) c(i) = c(irev(i))
    do l = 1, levels                    ! --- for each level in the FFT
       m = ishft(1, l)
       m2 = ishft(1, l - 1)
       do k = 0, n - 1, m               ! --- for each butterfly in a level
          do j = k, k + m2 - 1          ! --- for each point in a half bfly
             ce = exp(cmplx(0.0,(j - k) * -two_pi/real(m)))
             cr = ce * c(j + m2)
             cl = c(j)
             c(j) = cl + cr
             c(j + m2) = cl - cr
          end do
       end do
    enddo
 end subroutine fft
```

Productive Parallel 1D FFT ($n = 2^k$)

```
subroutine fft(c, n)
    implicit complex(c)
    dimension c(0:n-1), irev(0:n-1)
!HPF$ processors p(number_of_processors())
!HPF$ template t(0:n-1)
!HPF$ align c(i) with t(i)
!HPF$ align irev(i) with t(i)
!HPF$ distribute t(block) onto p
    two_pi = 2.0d0 * acos(-1.0d0)
    levels = number_of_bits(n) - 1
    irev = (/ (bitreverse(i,levels), i= 0, n-1) /)
    forall (i=0:n-1) c(i) = c(irev(i))
    do l = 1, levels                         ! --- for each level in the FFT
       m = ishft(1, l)
       m2 = ishft(1, l - 1)
       do k = 0, n - 1, m                    ! --- for each butterfly in a level
          do j = k, k + m2 - 1               ! --- for each point in a half bfly
             ce = exp(cmplx(0.0,(j - k) * -two_pi/real(m)))
             cr = ce * c(j + m2)
             cl = c(j)
             c(j) = cl + cr
             c(j + m2) = cl - cr
          end do
       end do
    enddo
 end subroutine fft
```

partitioning the j loop is driven by the data accessed in its iterations

Productive Parallel 1D FFT ($n = 2^k$)

```
subroutine fft(c, n)
    implicit complex(c)
    dimension c(0:n-1), irev(0:n-1)
!HPF$ processors p(number_of_processors())
!HPF$ template t(0:n-1)
!HPF$ align c(i) with t(i)
!HPF$ align irev(i) with t(i)
!HPF$ distribute t(block) onto p
    two_pi = 2.0d0 * acos(-1.0d0)
    levels = number_of_bits(n) - 1
    irev = (/ (bitreverse(i,levels), i= 0, n-1) /)
    forall (i=0:n-1) c(i) = c(irev(i))
    do l = 1, levels                        ! --- for each level in the FFT
       m = ishft(1, l)
       m2 = ishft(1, l - 1)
       do k = 0, n - 1, m                   ! --- for each butterfly in a level
          do j = k, k + m2 - 1              ! --- for each point in a half bfly
             ce = exp(cmplx(0.0,(j - k) * -two_pi/real(m)))
             cr = ce * c(j + m2)
             cl = c(j)
             c(j) = cl + cr
             c(j + m2) = cl - cr
          end do
       end do
    enddo
 end subroutine fft
```

partitioning the k loop is subtle: driven by partitioning of j loop

partitioning the j loop is driven by the data accessed in its iterations

Productive Parallel 1D FFT ($n = 2^k$)

```
subroutine fft(c, n)
    implicit complex(c)
    dimension c(0:n-1), irev(0:n-1)
!HPF$ processors p(number_of_processors())
!HPF$ template t(0:n-1)
!HPF$ align c(i) with t(i)
!HPF$ align irev(i) with t(i)
!HPF$ distribute t(block) onto p
    two_pi = 2.0d0 * acos(-1.0d0)
    levels = number_of_bits(n) - 1
    irev = (/ (bitreverse(i,levels), i= 0, n-1) /)
    forall (i=0:n-1) c(i) = c(irev(i))
    do l = 1, levels                        ! --- for each level in the FFT
       m = ishft(1, l)
       m2 = ishft(1, l - 1)
       do k = 0, n - 1, m                   ! --- for each butterfly in a level
          do j = k, k + m2 - 1              ! --- for each point in a half bfly
             ce = exp(cmplx(0.0,(j - k) * -two_pi/real(m)))
             cr = ce * c(j + m2)
             cl = c(j)
             c(j) = cl + cr
             c(j + m2) = cl - cr
          end do
       end do
    enddo
 end subroutine fft
```

partitioning the k loop is subtle: driven by partitioning of j loop

stride is problematic for polyhedral methods

partitioning the j loop is driven by the data accessed in its iterations

Some Lessons from HPF

- **Good parallelizations require proper partitionings**
  - —inferior partitionings will fall short at scale
- **Excess communication undermines scalability**
  - —both frequency and volume must be right!
- **Must exploit what smart users know**
  - —allow the power user to hide or avoid latency
- **Single processor efficiency is critical**
  - —node code must be competitive with serial versions
  - —must use caches effectively
- **Abstraction is good in moderation**
  - —compilation challenges for abstract models can sometimes be daunting

Challenges of Exascale Hardware

- Complexity
- Concurrency
- Scale
- Heterogeneity
  - —architecture
  - —performance
- Failure and resilience
- Power
  - —focus: maximize locality to minimize data movement

Some Exascale Technology Needs

- Programming models, compilers, runtime systems
  - —communication
    - – point-to-point, collective, near neighbor, ...
  - —synchronization
    - – ordering, mutual exclusion, producer consumer
  - —partitioning
  - —placement
  - —scheduling
- Tools ecosystem

A Hierarchy of Programming Models

- **Domain specific languages**
  - **—e.g., TCE, SPIRAL**
- **Frameworks**
  - **—e.g., Chombo**
- **Programming languages**
- **Libraries**

Programming Models for the Exascale

- MPI + X is the front runner
- MPI role at exascale [“MPI at Exascale”, Thakur, Scidac 2010]
  - “MPI being used to communicate between address spaces”
  - “use some other shared-memory programming model (OpenMP, UPC, CUDA, OpenCL) for programming within an address space”
- Why not just X?
  - skeptic: but MPI provides all the things I know and love
    - communicators for processor subsets
    - collectives across communicators
  - PGAS model can provide those directly instead
    - ... along with compiler support to make it easier to use!

Example: Coarray Fortran 2.0

- Teams: process subsets, like MPI communicators
  - formation using team_split (like MPI_Comm_split)
  - collective communication
- Topologies
- Coarrays: shared data allocated across processor subsets
  - declaration: double precision :: a(:,:)[*]
  - dynamic allocation: allocate(a(n,m)[@row_team])
  - access: x(:,n+1) = x(:,0)[p] *(p is a rank in the "default team")*
- Latency tolerance
  - hide: predicated asynchronous copy, asynchronous collectives
  - avoid: function shipping
- Synchronization
  - event variables: point-to-point sync; async completion
  - finish: SPMD construct inspired by X10
- Copointers: structured pointers to distributed data (in progress)
- Multithreading: compiler and runtime support for work stealing (in progress)
- Accelerated computing: map loop nests (semi-)automatically to manycore (planned)

Scalable PGAS Programming Model

Issues (see "MPI at exascale," Thakur, SciDAC 2010)

- Scalable bookkeeping state
  - — maintain little global state per "process"
    - avoid full knowledge of processor subsets
  - — CAF 2.0 team construction applied to MPI
    - "Exascale Algorithms for Generalized MPI_Comm_Split" [Moody et al. EuroPar 11]
- Very little memory management within MPI
  - — all memory for communication can be in user space
  - — consistent with PGAS models
- Collectives are useful, scalable, and efficient
- "Some parts of MPI are being fixed for exascale" (MPI-3)
  - — RMA
  - — non-blocking and (maybe) neighborhood collectives

Mapping to Heterogeneous Nodes

- Explicit programming: CUDA, OpenCL?
  - too low level and detailed
- Today: Cray's accelerator pragmas [Levesque, SciDAC 2011]
  - ```
    !$omp acc_region_loop private(...)
    !$omp acc_data acc_copyin(...)
        ...
    !$omp end acc_region_loop
    ...
    !$omp acc_update host(x)
    ...
    !$omp acc_update acc(x)
    !$omp acc_data present(...)
    ```
  - benefits: handle detailed synthesis of code for manycore
- Future: preference for more declarative pragmas, if any
  - leverage type system: constant variables can be "copyin"
- Challenge: semi-automatically mapping complex codes
  - managing irregular data, handling dependences, ...
    ```

PGAS Data Models at Scale

- **Distributed state**
- **Distributed descriptors**
- **Scalable data movement**
- **Scalable synchronization**
- **Emerging issue: fault tolerance**
  - **persistance**
  - **recoverability**

- **Approach: all members of a team do the following ...**
  - **agree on a handle**
  - **allocate a piece of the data**
  - **data movement and synchronization: point-to-point or collective**

Support for Coupling - I

Location service

— locate a component by name, e.g. “ocean simulation component”

- returns a handle, and an identifier for a node

— service must be distributed for scalability

— fault tolerance: no single point of failure

- service implementation could use replication

Support for Coupling - II

Scalable binding

- example: CESM
  - model coupler must bind to ocean and atmosphere components
  - use a handle from a registry to arrange for scalable communication with each component
    - establish appropriate many-many, many-one, or one-many mapping between corresponding ranks in coupler and target component
- fault tolerance
  - log communication through a binding
  - notice when a binding disappears
  - be able to re-establish a binding using location service

Locality-aware Dynamic Scheduling

- Issues
  - — incoming work from function shipping
  - — critical path
- Approaches
  - — need scalable, locality-aware, priority-aware strategies
  - — rethink data structures, e.g. recursive array layouts
  - — support affinity hints
  - — rethink dynamic scheduling decomposition
    - e.g., use traversal orders derived from space filling curves for hierarchical locality
  - — provide support for reordering data and computation for irregular problems
    - explicitly represent schedules for irregular work
    - recompute schedules on demand, e.g. periodic sorting
    - reuse schedules to amortize overhead
  - — tighter integration with HW

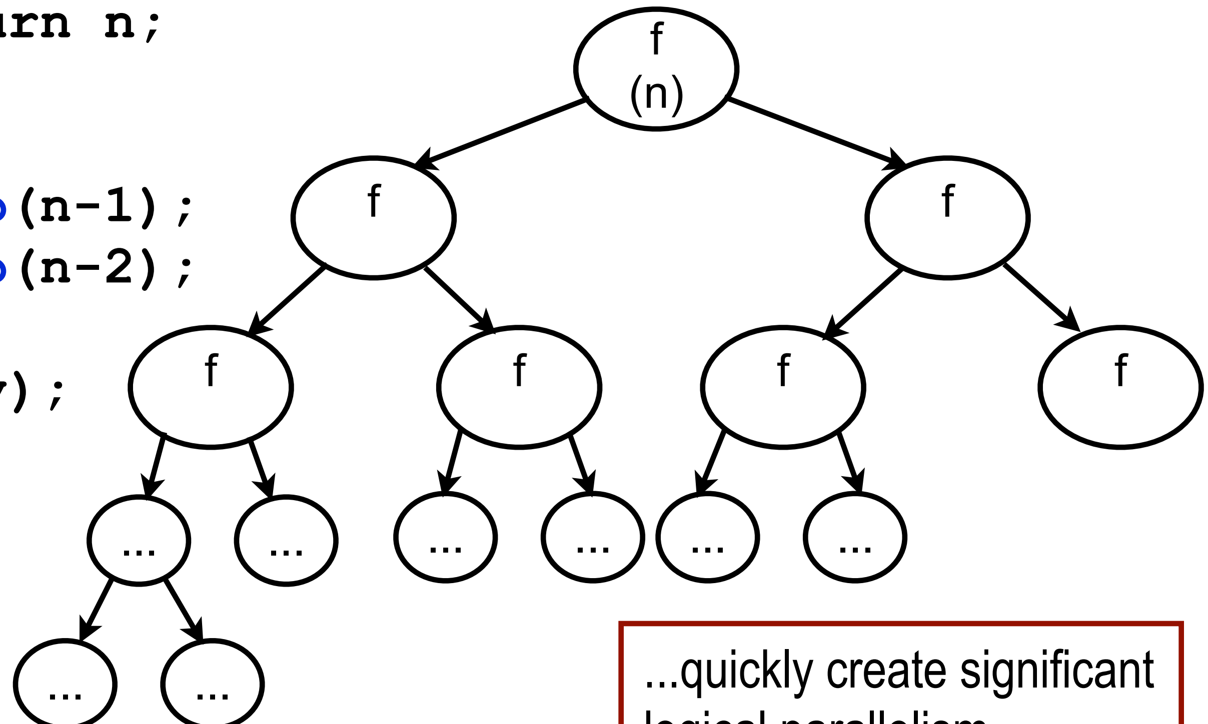
Supporting the Tools Ecosystem

- **Performance tools will be extremely important for the exascale**
- **Pinpoint and quantify power consumption for tuning**
- **Pinpoint inefficiencies**
  - — **insufficient parallelism**
  - — **power consumption**
  - — **data movement**
  - — **overhead**

Cilk: A Multithreaded Language

```
cilk int fib(n) {
  if (n < 2) return n;
  else {
    int x, y;
    x = spawn fib(n-1);
    y = spawn fib(n-2);
    sync;
    return (x + y);
  }
}
```

<u>asynchronous</u> calls create logical tasks that only block at a **sync**...



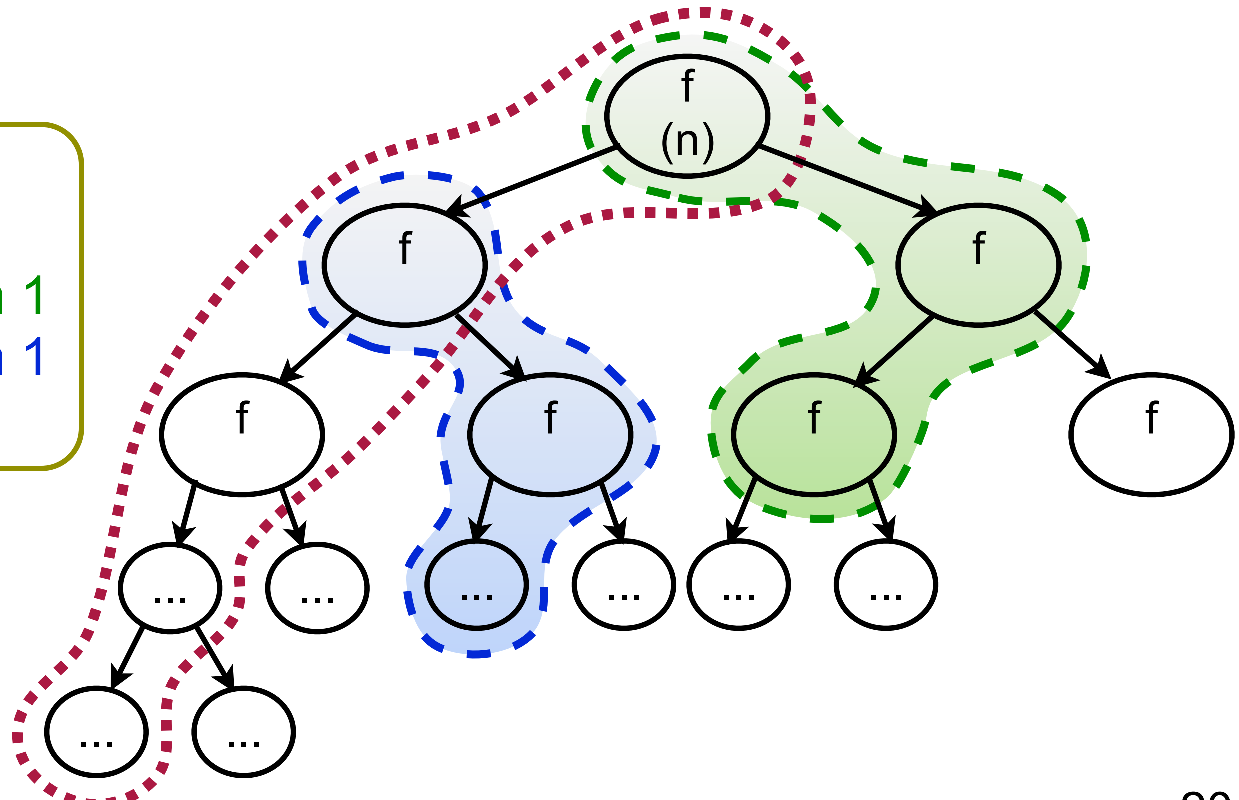
...quickly create significant logical parallelism.

Cilk Program Execution using Work Stealing

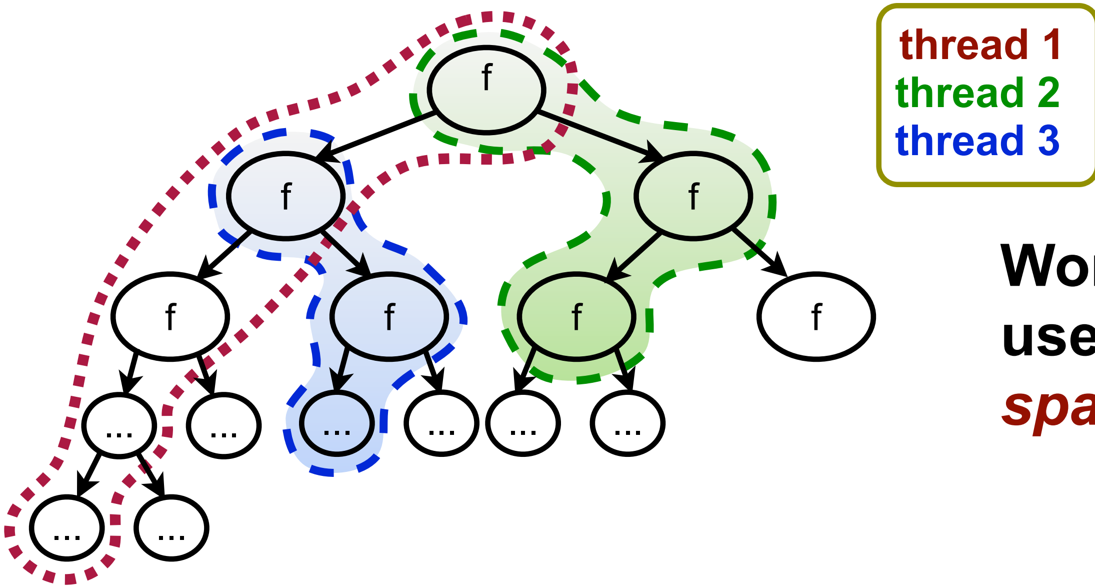
- **Challenge: Mapping logical tasks to compute cores**
- **Cilk approach:**
  - — **lazy thread creation plus work-stealing scheduler**
    - **`spawn`: a potentially parallel task is available**
    - **an idle thread steals tasks from a random working thread**

Possible Execution:

thread 1 begins
thread 2 steals from 1
thread 3 steals from 1
etc...

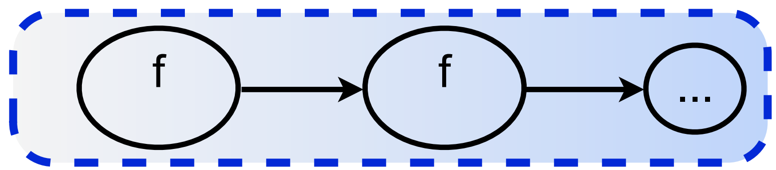


Call Path Profiles with Work Stealing

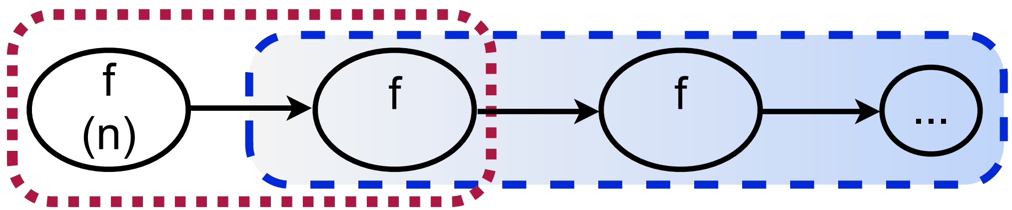


Work stealing *separates* user-level calling contexts in *space and time*

- **Consider thread 3:**
  - **physical call path:**



  - **logical call path:**



Logical call path profiling: Recover *full* relationship between *physical* and *user-level* execution

Attributing Costs: Blame Shifting

- **Problem: in many circumstances sampling measures symptoms of performance losses rather than causes**
  - worker threads waiting for work
  - threads waiting for a lock
  - MPI process waiting for peers in a collective communication
- **Approach: shift blame for losses from victims to perpetrators**
  - who is failing to shed parallel work to keep everyone busy
  - who is holding the lock and stalling others
  - who is delaying progress at a collective call site
- **Flavors**
  - analysis only
  - active measurement

Barriers to Adopting New Models

- **Application codes are long lived**
  - **must run on several generations of architecture**
- **Developers are conservative**
  - **want to use standard languages**
- **Moving forward ...**
  - **work with language standards committee to add new features**