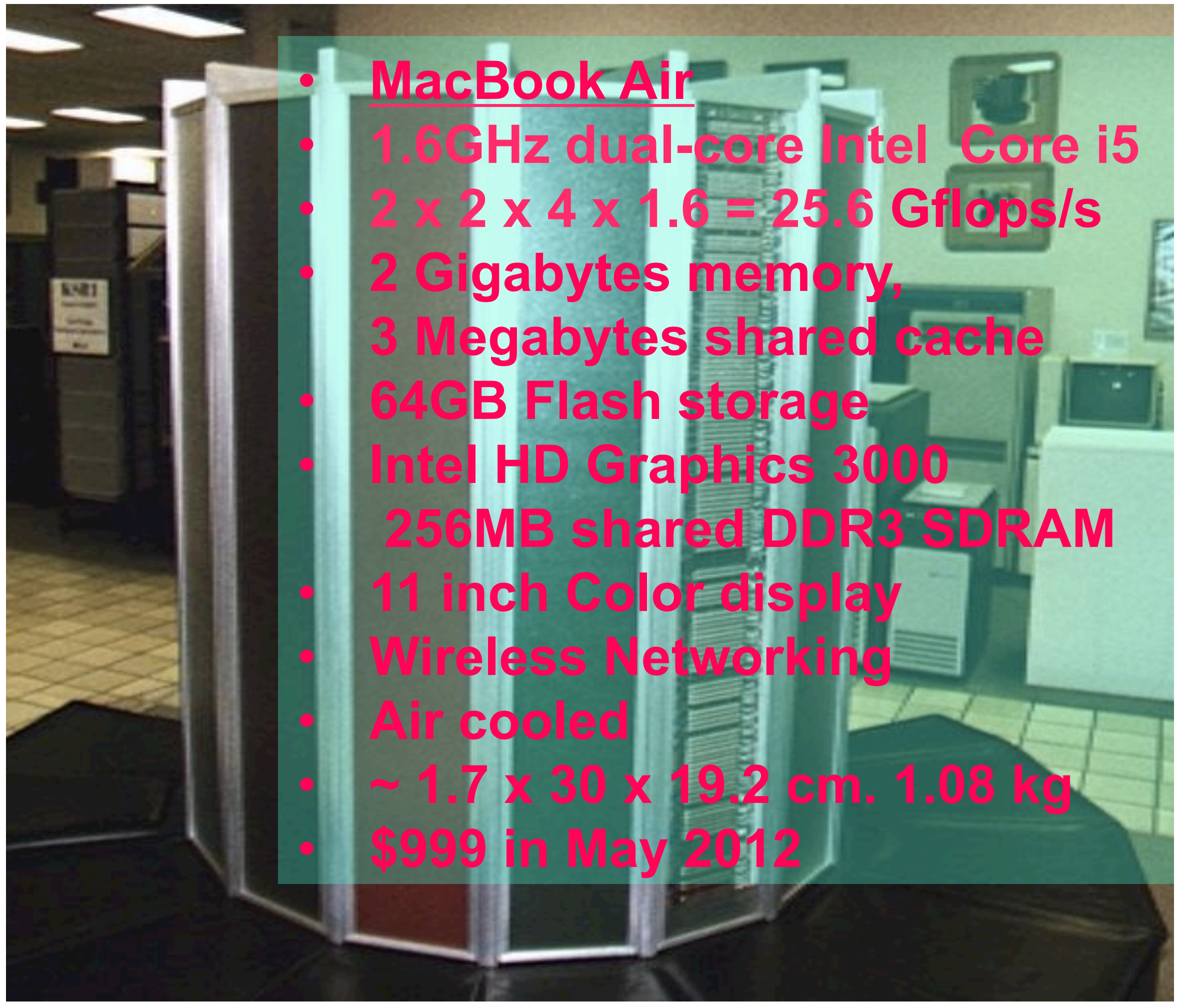
Computing at a million laptops per second

Scott B. Baden
Dept. of Computer Science and Engineering
University of California, San Diego

Today's Laptop is Yesterday's Supercomputer

- Cray-1 Supercomputer
- 80 MHz processor
- 240 Mflops/sec
- 8 Megabytes memory

- Water cooled
- 1.8m H x 2.2m W
- 4 tons
- Over $10M in 1976



- MacBook Air
- 1.6GHz dual-core Intel Core i5
- 2 x 2 x 4 x 1.6 = 25.6 Gflops/s
- 2 Gigabytes memory,
  3 Megabytes shared cache
- 64GB Flash storage
- Intel HD Graphics 3000
  256MB shared DDR3 SDRAM
- 11 inch Color display
- Wireless Networking
- Air cooled
- ~ 1.7 x 30 x 19.2 cm. 1.08 kg
- $999 in May 2012

Supercomputer Evolution

- Performance increases ×1000 every 10 years

240 MFlops	1TFlop	1 PFlop	10.5 PFlops
1976	1996	2008	2012
1 core	4510	122,400	705,024
115KW	850KW	2.35 MW	9.89MW
2KF/W	1MF/W	0.4GF/W	1GF/W

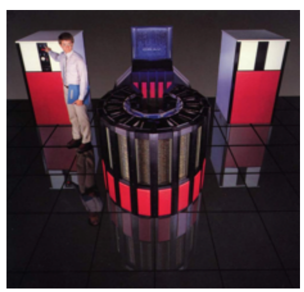
Cray-1

ASCI Red

Roadrunner

K Computer
×410,000 MacBook Air

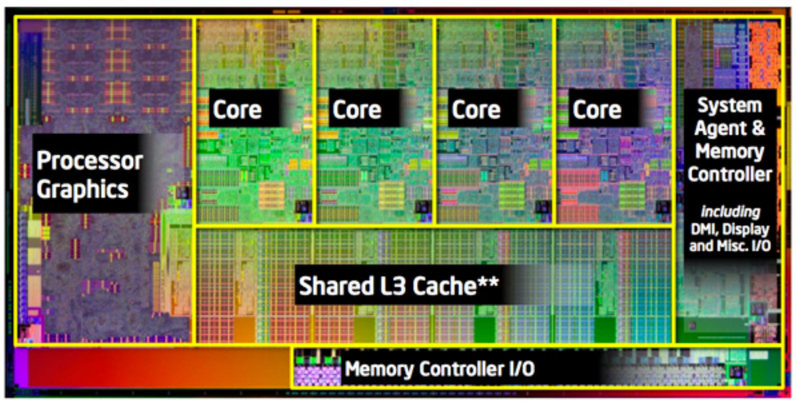
Cray-2
1985-90, 1.9 GFlops
195 KW (10 KF/W)
Ipad-2 today (5W)

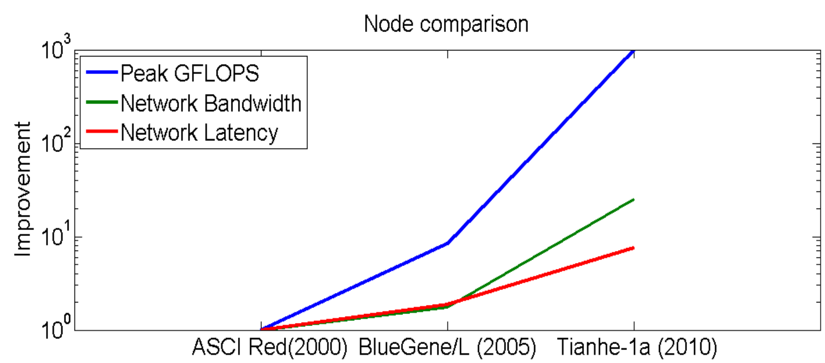


Lindgren
@pdc.kth.se
2011, 305 TFlops
36384 cores
600 KW
35,000 Ipads
×12,000 MacBook Air

Technological trends

- Growth: #cores/socket
- Memory/core will shrink
- Complicated software-managed parallel memory hierarchy
- Communication bandwidth growing slowly, greater variation in delays





Intel Sandybridge, anandtech.com

Consequences for the programmer

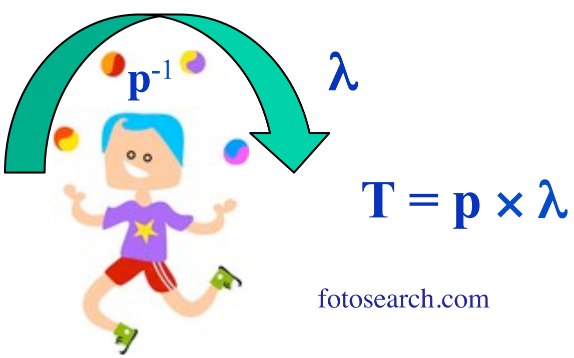
- Performance programming gets more complicated
  - Increase the number of ops per word moved
  - Hide or avoid communication
- Domain specific knowledge plays a crucial role in optimizing performance
- Need a way to apply expert knowledge without having to become an expert
  - Custom translation
  - Embedded Domain Specific Languages

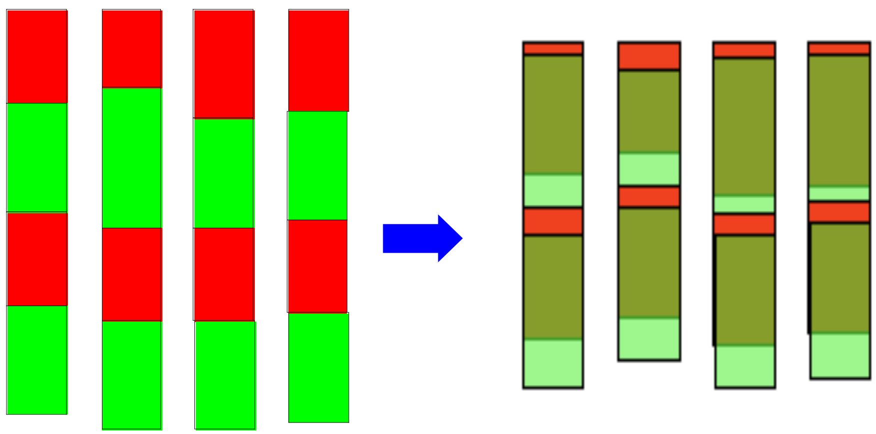
Roadmap

- Automatically restructure conventional code using a custom source-to-source translator ...
- … that captures semantic knowledge of the application domain … thereby improving performance
- Embedded Domain Specific Languages
  - Automatically tolerate communication delays
  - Squeeze out library overheads
- Library primitives → primitive language objects

Tolerating communication

- Overlap communication with computation
  - Split phase algorithms
  - Scheduling
  - Over decomposition to cope with Little's law
- Implementation policies entangled with correctness issues
  - Non-robust performance
  - High development costs





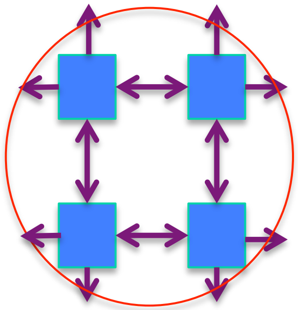
Bamboo Source-to-Source Translator

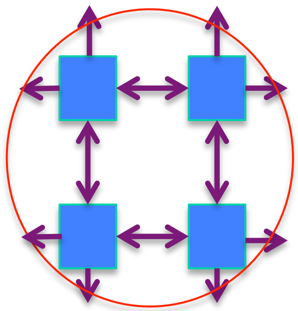
- Deterministic procedure for translating MPI code to a dataflow formulation, using dynamic and static information about MPI call sites
- Custom, domain-specific translator
  - Understands the MPI API
  - Targets the Tarragon library [Pietro Cicotti '11]
  - Built using ROSE
  - Tan Nguyen (UCSD)
  - Eric Bylaska (PNNL), Daniel Quinlan (LLNL), John Weare (UCSD)

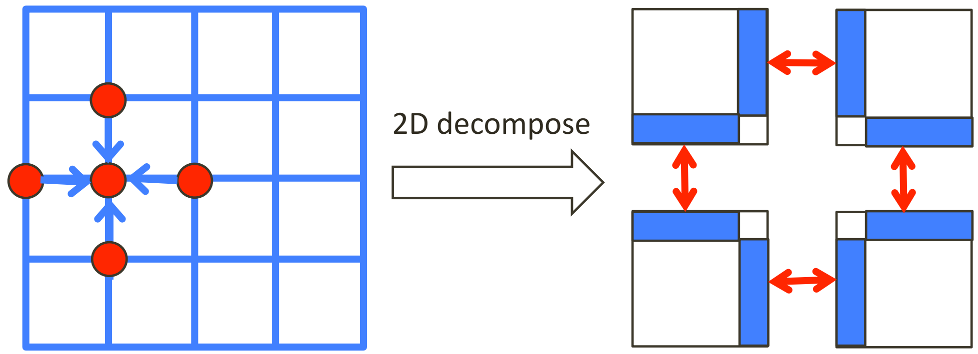
Example: MPI with annotations

Calculate indices for left/right/up/down

```
#pragma bamboo olap (nearest_neighbor)  {
 for (it=0; it<number_of_iterations; it++)
     #pragma bamboo send{
        pack outgoing ghost cells
        MPI_Isend(SendGhostcells,left/right/up/down)
     }
     #pragma bamboo receive{
        MPI_Recv(RecvGhostcells,left/right/up/down)
        unpack incoming ghost cells
     }
     MPI_Waitall();
     for(j=1; j < N/numprocs -1; j++)
        for(i=1; i < N -1; i++)
           V(j,i) = c*(U(j,i+1) + U(j,i-1) + U(j+1,i) + U(j-1,i));
   swap(U, V);
   }
}
```





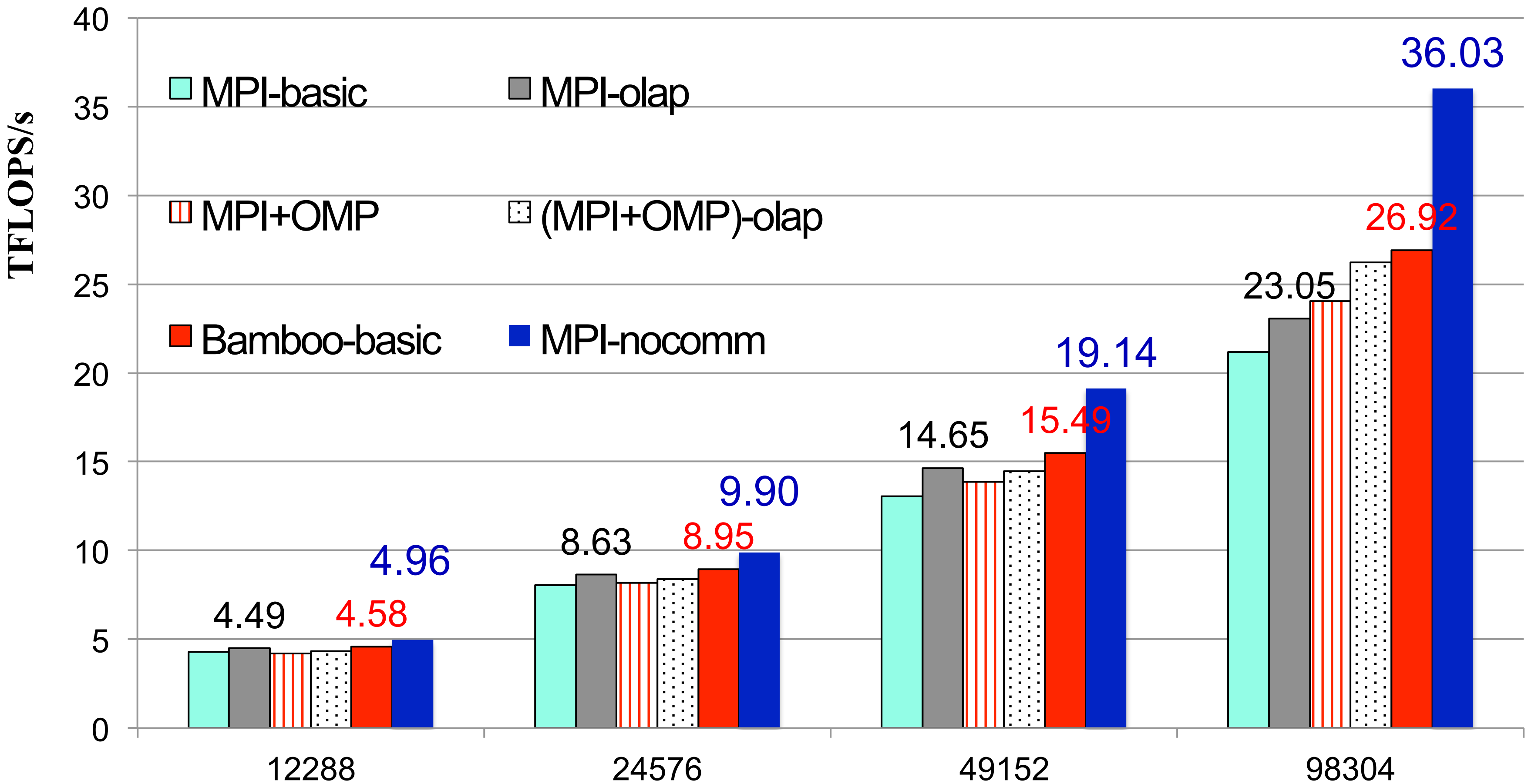


Performance

- Cray XE-6 at NERSC (Hopper)
- 153,216 cores; dual socket 12-core Magny Cours
- 4 NUMA nodes per Hopper node, each with 6 cores
- 3D Toroidal Network
- Gnu 4.6.1, -O3 –ffast-math
- ACML 4.4.0
- 2 Application Motifs
  - Stencil Method
  - Dense linear algebra

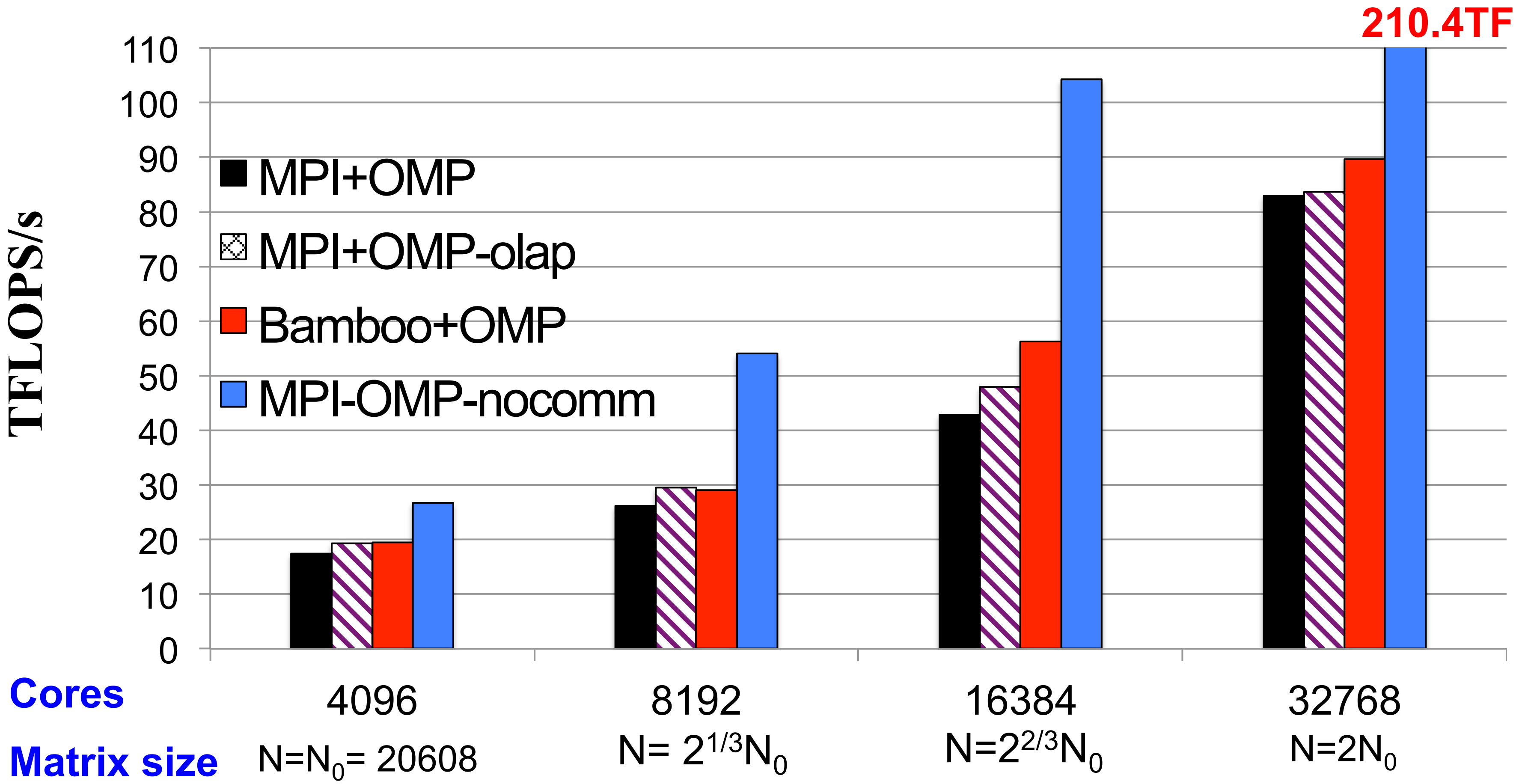
Stencil application performance

- Solve 3D Laplace equation, Dirichlet BCs (N=3072^3)
  7-point stencil $\Delta u = 0$, $u=f$ on $\partial\Omega$
- Added 4 Bamboo pragmas to a 419 line MPI code

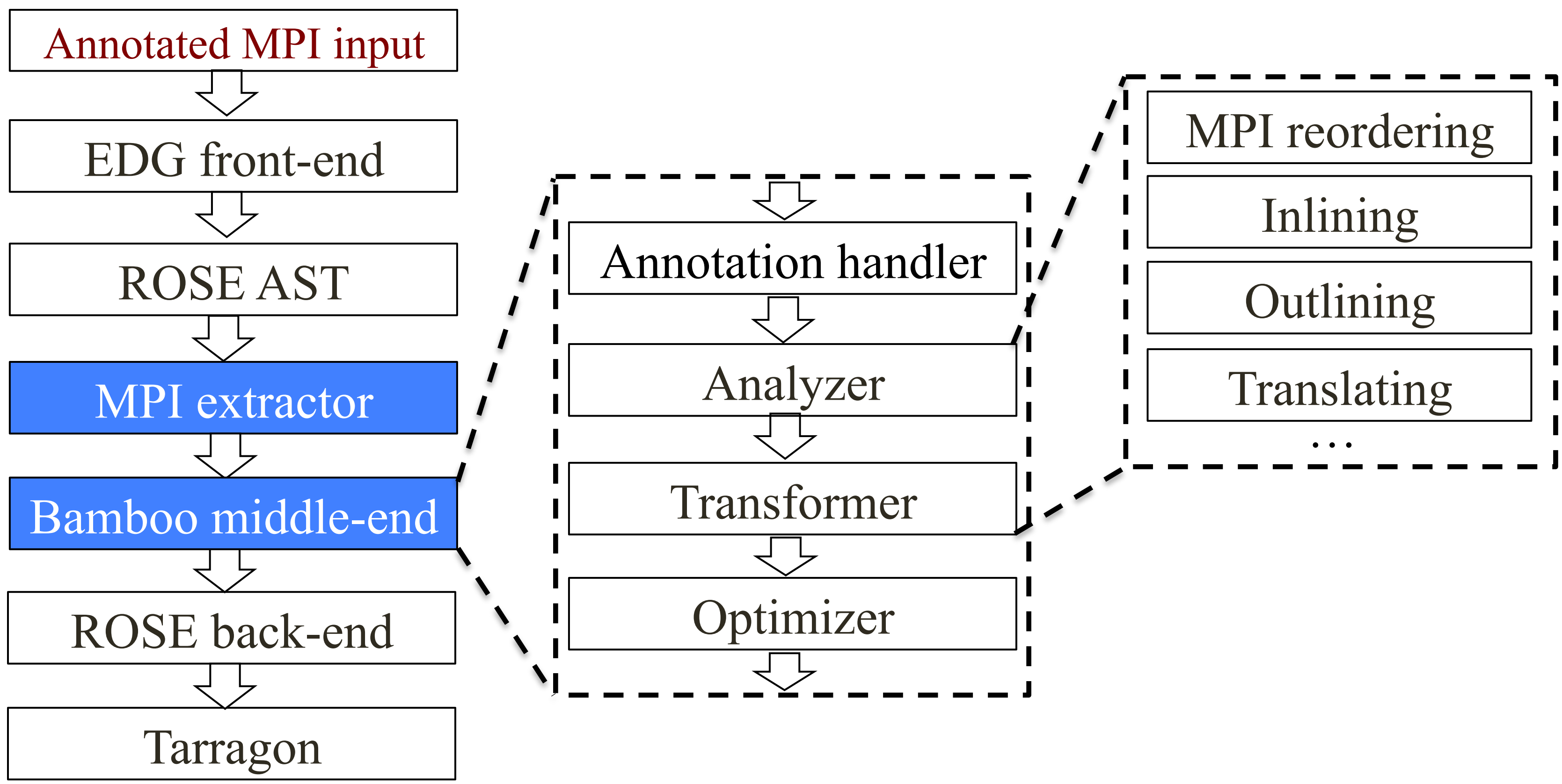


Communication Avoiding Matrix Multiplication

- Pathological matrices in Planewave basis methods for *ab-initio* molecular dynamics ($N_g^3 \times N_e$), For Si: N_g=140, Ne=2000
- Weak scaling study, used OpenMP, 23 pragmas, 337 lines



Bamboo Translator



Discussion and Related work

- The invocation of an MPI program defines a dataflow graph that we may construct via domain-specific translation & thereby improve performance by hiding communication
- Related work
  - Marjanovic ['10] task parallelism via OpenMP annotations (MPI/SMPSs)
  - Charm++, AMPI [Kalé '93] Virtualization, RMI on global name space
  - Danalis ['05] – transform MPI to realize overlap in collectives
  - Preiss et al. Analysis and optimization of global communication patterns ['10]

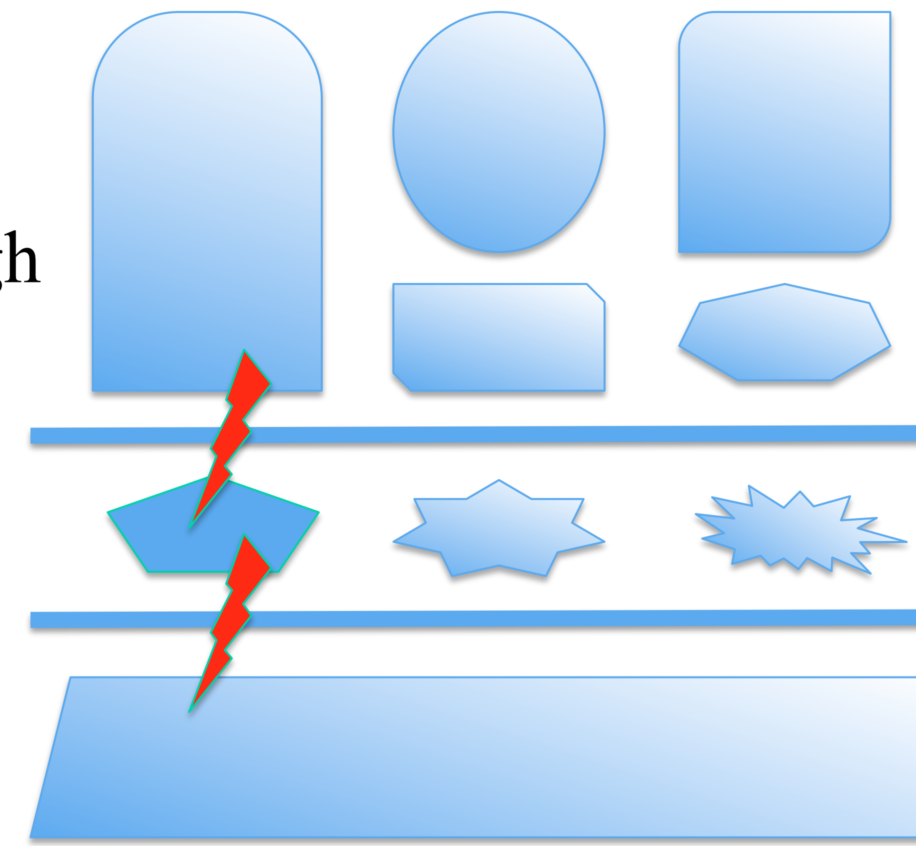
Road Map

- Two performance programming problems treat via embedded DSLs
  - Hide communication delays in massively parallel MPI applications
  - Reducing abstraction overheads in libraries

Languages vs. Libraries

- Languages
  - A few simple constructs
  - Compiler support
- Libraries
  - Specialization through layering
  - Run time support

These transitions are expensive



Overheads in an array class library

- Array indexing: `A(x,y) = B(x,y) + k*C(x,y)`

```
Array A(M,N), B(M,N), C(M,N)
A.At(x,y) = B.At(x,y) + k * C.At(x,y)
```

vs

```
idx = x + y * N;
A[idx] = B[idx] + k * C[idx]
```

- Opportunities for optimization
  - A, B and C have conforming shapes: avoid redundant computation of index offset: `(x+ y*N)`
  - Stride of A `(N)` can be determined from constructor call site, or cached on the stack
- Compilers can optimize away the overheads for variables on the stack, but not on the heap

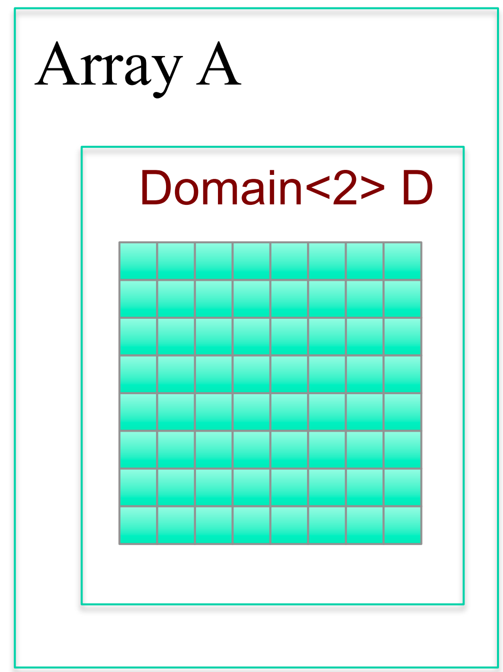
  Gnu 4.4, Intel 12.0.2

Saaz

- Array class library for analyzing turbulent flow data on structured meshes, includes CFD query functionality
- Arrays have *configurations: e.g. row vs. column major*
- Domain abstraction (FIDIL, KeLP, Titanium)

```
Domain<2> dmn(xmin, xmax, ymin, ymax)
Array A(dmn, Layouts::RowMajor)
foreach p in dmn
    A[p] = B[p] + k*C[p]
```

- Invariants
  - Domains are constant
  - Layouts and other configurations are constant
- Constraints
  - No windowing; array aliases to whole arrays only



Overheads and how to eliminate them

- Data organization details have been abstracted away to improve programmability
- Compilers don't understand the data structures and can't peer behind the object interface
- Refinement based white box optimization
  - Infer specializations of the array type from context
  - Expose library semantics to the translator

Domain Specific Translation with Tettnang

- Reduce Saaz overheads by ×20, competes with C hand coding
  - Resolve, at compile time, certain control flow decisions that impact performance, using knowledge about the library
  - Inlining that peers inside object interface
- Thesis topic of Alden King ['12]
  - Sutanu Sarkar and Eric Arobone (UCSD MAE)
  - Johan Hoffman and Niclas Jansson (KTH)
- Built with Rose (Quinlan et al.)

Tettnang

- Multi-pass translator
- Type refinement
  General array → row major order array .. With an index origin of zero and with axes numbered (0,2,1)
- Attributes fixed at construction time establish
  - Equivalence between object members and constructor parameters
  - Control flow paths
- Transformations
  - Inlining
    - Replace member function calls by values passed at construction time
  - Statically resolve control flow
    - Remove guards and untaken branches
    - Replace *p→( ) with an equivalent expression, e.g, body of a suitably chosen function referred to by the pointer

Benefits of Tettnang's optimizations

- If we know the layout (RowMajor), then we can use the actual constructor arguments to generate the array indices

```
Domain dmn(xmin, xmax, ymin, ymax);
Array A(dmn, Layouts::RowMajor);          A[i] += B[i]
Array B(dmn, Layouts::RowMajor);

unsigned long long off = i.x * (ymax – ymin + 1) + i.y;
A.dest->datap[off] += B.dest->datap[off]
```

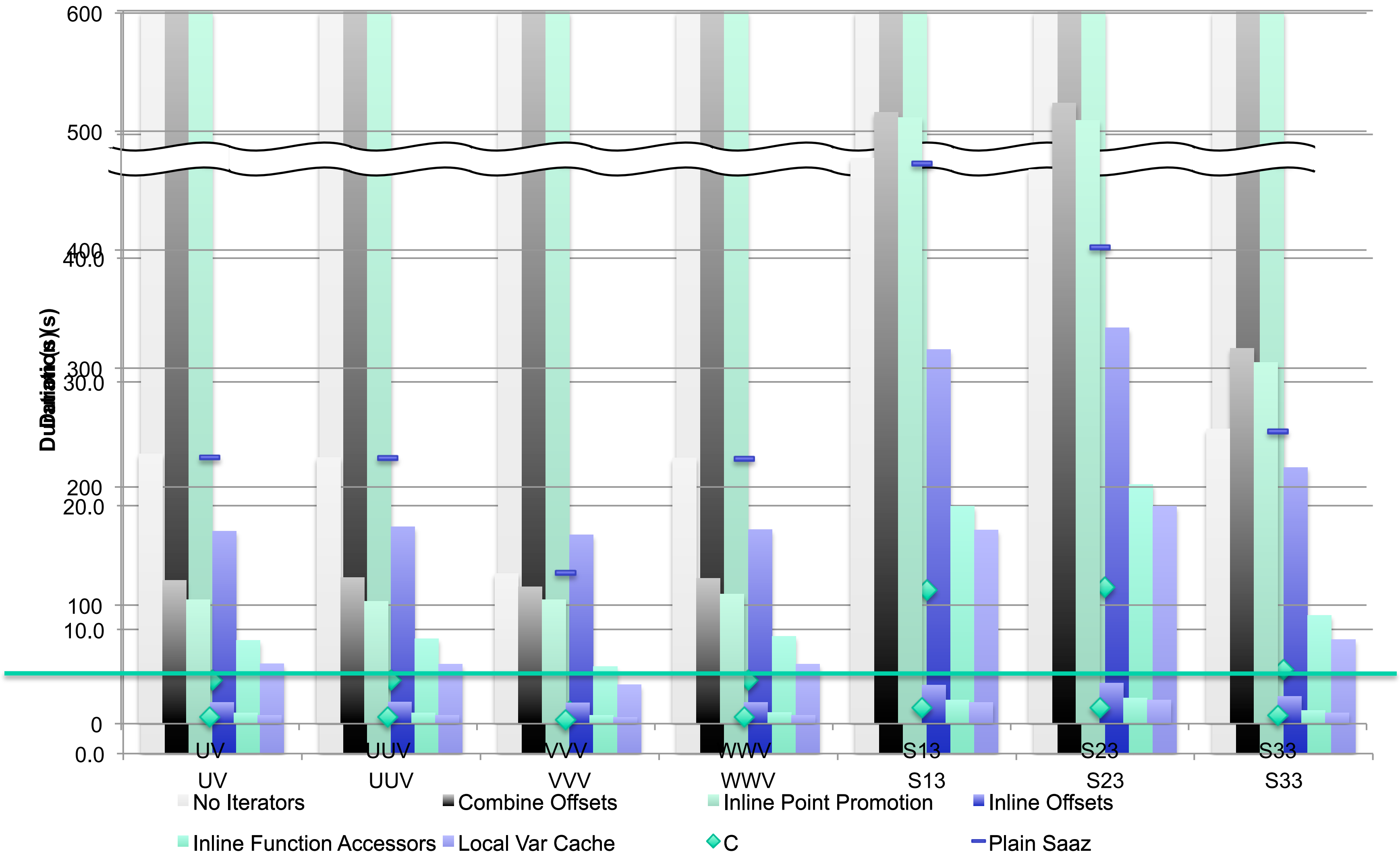
- If not, we have to query the array object for that information (expensive), e.g traditional compilers for heap variables

```
A.dest->datap[A.LinearIdx(i)] += B.dest->datap[B.LinearIdx(i)]
If (A.layout == Layouts::RowMajor)
    off = i.x * A.Domain().Length(1) + i.y;
else offs = i.x + i.y*A.Domain().length(0)
A.dest->datap[off] += B.dest->datap[off]
```

Results

- 8 Saaz queries, including…
  - <wwv> lateral transport of vertical fluctuations
  - <s13> dissipation in the x-y plane
    (~ likelihood of breakup and separation)
- Testbed
  - 2x Intel Xeon 4 core Harpertown
    - Single threaded tests
  - 32 GB RAM
  - Minimal timings of 3-10 runs
  - Datasets: 4 variables on a mesh: 1535 × 768 × 768
  - Gnu 4.4

Tettnang improves performance dramatically



Discussion and Related Work

- Convenient to encode type refinement information in an object's members
- We can avoid interpretation overheads, via domain specific translation, recovering type refinement information
- Expression templates
  - Can' handle optimize functional composition, but not across statements (unless we overload ',' operator)
  - Difficult to use, not suitable for application focused users
- Delayed evaluation [Dryad, DESOBLAS]
- Dynamic compilation [Noel, '96]
- Value profiling [Calder '99, Aigner '96]
- Rapid Type Analysis [Bacon '96, Calder '99]: virtual functions, not pointers, Tettnang performs more detailed type refinements
- A++/P++ [Quinlan '06] – annotations
- Telescoping Languages: use semantic knowledge, static typing of dynamic languages [Kennedy '05]

Conclusions

- Domain specific translation treats user defined abstractions as first class language objects
  - Latency hiding
  - Remove the overheads of abstraction
  - Heterogeneous processing and optimizations

  Mint, Didem Unat, PhD 2012
  sites.google.com/site/mintmodel
- Important role in exascale computing, or whatever we have in our laptop
- Bamboo and Tettnang available soon

Acknowledgements and Support

- Eric Arobone, Eric Bylaska (PNNL), Xing Cai (Simula), Pietro Cicotti ['11], Han Suk Kim ['11], Sherry Li (LBNL), Dan Quinlan (LLNL), Sutanu Sarkar, Ross Walker (SDSC), John Weare
- My group's web site: www-cse.ucsd.edu/groups/hpcl/scg