TAU Parallel Performance System

DOD UGC 2004 Tutorial

Part 3: TAU Applications and Developments

Tutorial Outline – Part 3

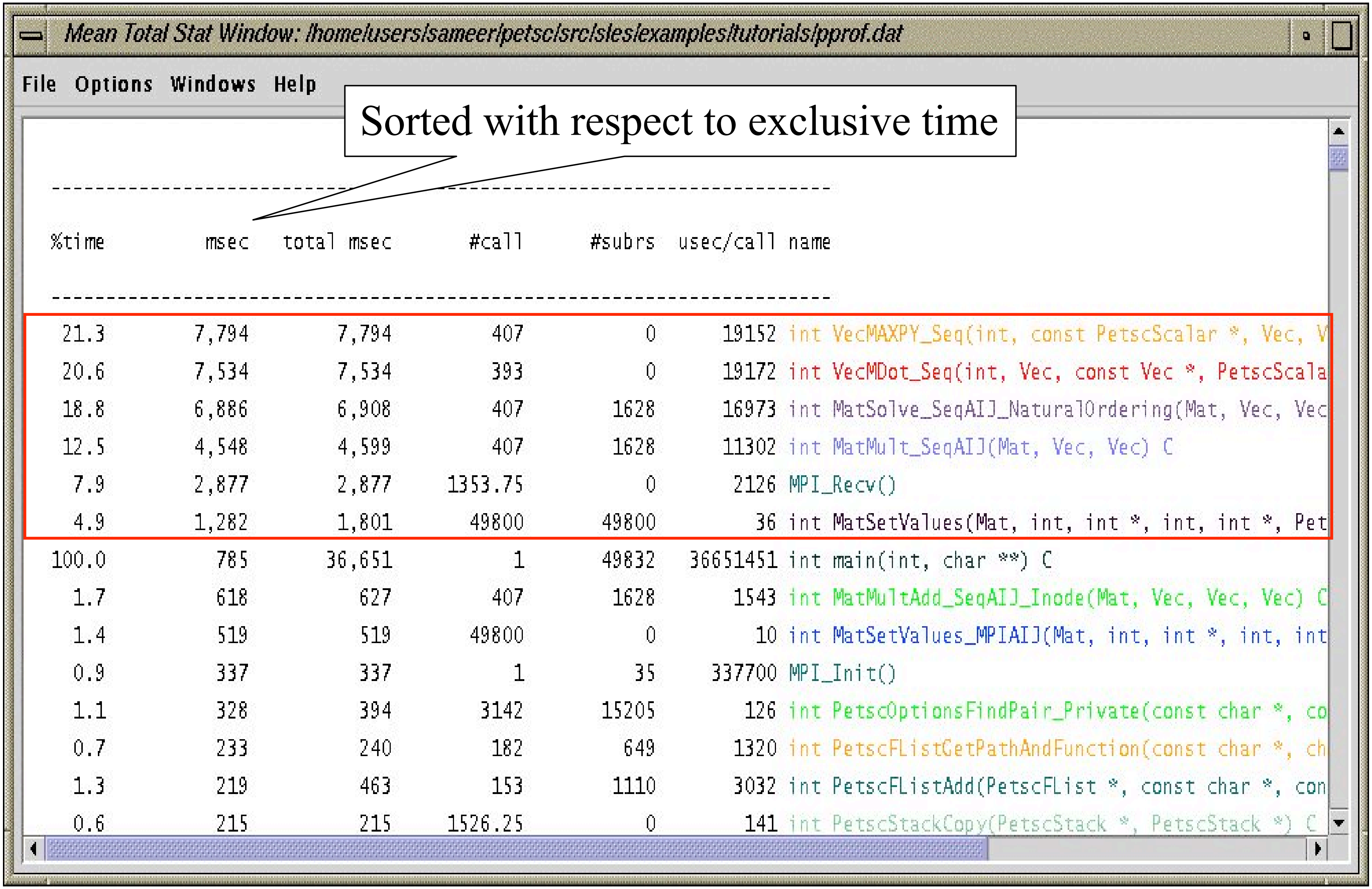
TAU Applications and Developments

- Selected Applications
  - PETSc, EVH1, SAMRAI, Stommel
  - mpiJava, Blitz++, SMARTS
  - C-SAFE/Uintah
  - HYCOM, AVUS
- Current developments
  - PerfDMF
  - Online performance analysis
  - ParaVis
- Integrated performance evaluation environment

Case Study: PETSc v2.1.3 (ANL)

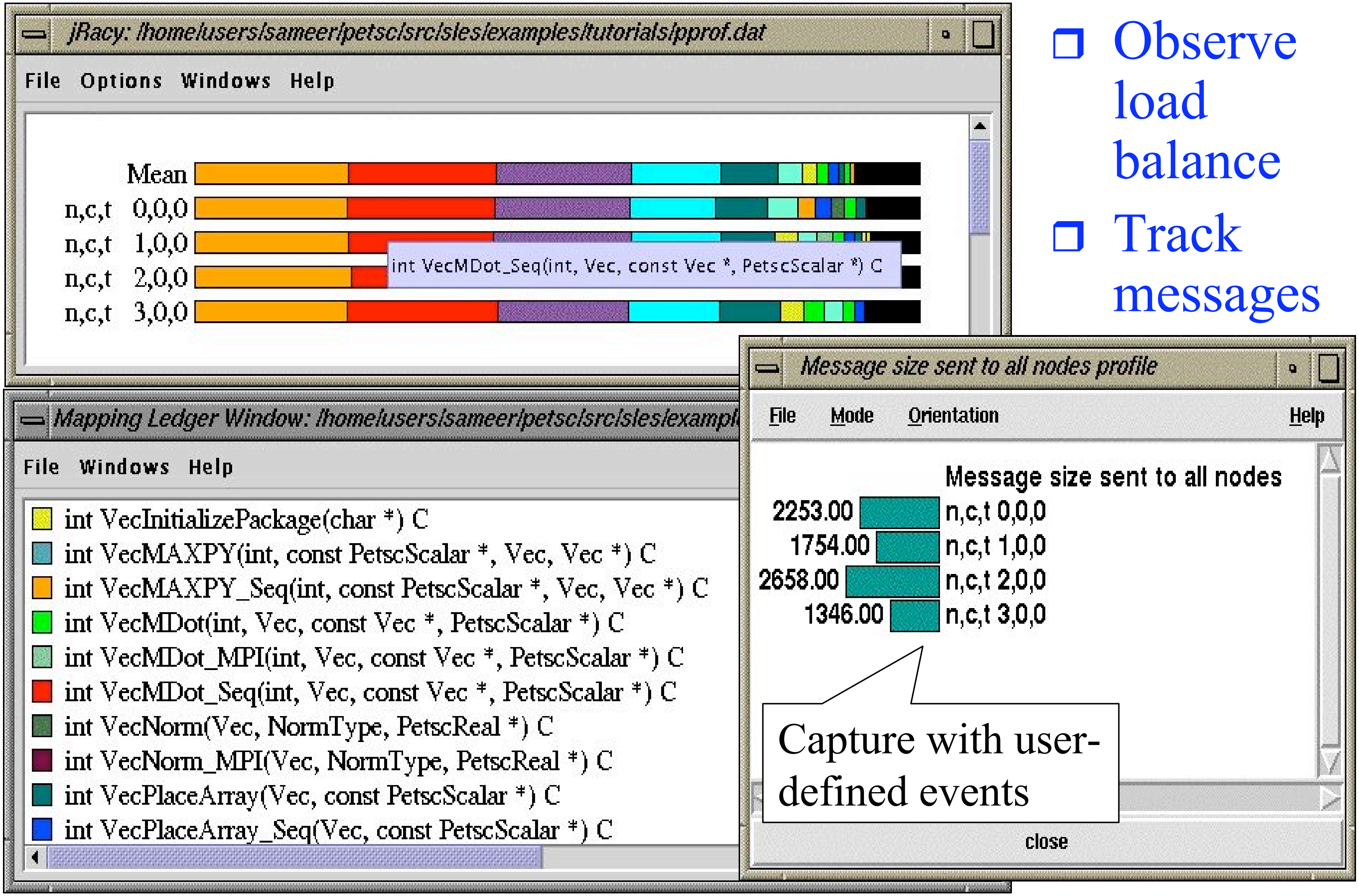
- Portable, Extensible Toolkit for Scientific Computation
- Scalable (parallel) PDE framework
  - Suite of data structures and routines (374,458 code lines)
  - Solution of scientific applications modeled by PDEs
- Parallel implementation
  - MPI used for inter-process communication
- TAU instrumentation
  - PDT for C/C++ source instrumentation (100%, no manual)
  - MPI wrapper interposition library instrumentation
- Example
  - Linear system of equations (Ax=b) (SLES) (ex2 test case)
  - Non-linear system of equations (SNES) (ex19 test case)

PETSc ex2 (Profile - wallclock time)



%time	msec	total msec	#call	#subrs	usec/call	name
21.3	7,794	7,794	407	0	19152	int VecMAXPY_Seq(int, const PetscScalar *, Vec, V
20.6	7,534	7,534	393	0	19172	int VecMDot_Seq(int, Vec, const Vec *, PetscScala
18.8	6,886	6,908	407	1628	16973	int MatSolve_SeqAIJ_NaturalOrdering(Mat, Vec, Vec
12.5	4,548	4,599	407	1628	11302	int MatMult_SeqAIJ(Mat, Vec, Vec) C
7.9	2,877	2,877	1353.75	0	2126	MPI_Recv()
4.9	1,282	1,801	49800	49800	36	int MatSetValues(Mat, int, int *, int, int *, Pet
100.0	785	36,651	1	49832	36651451	int main(int, char **) C
1.7	618	627	407	1628	1543	int MatMultAdd_SeqAIJ_Inode(Mat, Vec, Vec, Vec) C
1.4	519	519	49800	0	10	int MatSetValues_MPIAIJ(Mat, int, int *, int, int
0.9	337	337	1	35	337700	MPI_Init()
1.1	328	394	3142	15205	126	int PetscOptionsFindPair_Private(const char *, co
0.7	233	240	182	649	1320	int PetscFListGetPathAndFunction(const char *, ch
1.3	219	463	153	1110	3032	int PetscFListAdd(PetscFList *, const char *, con
0.6	215	215	1526.25	0	141	int PetscStackCopy(PetscStack *, PetscStack *) C

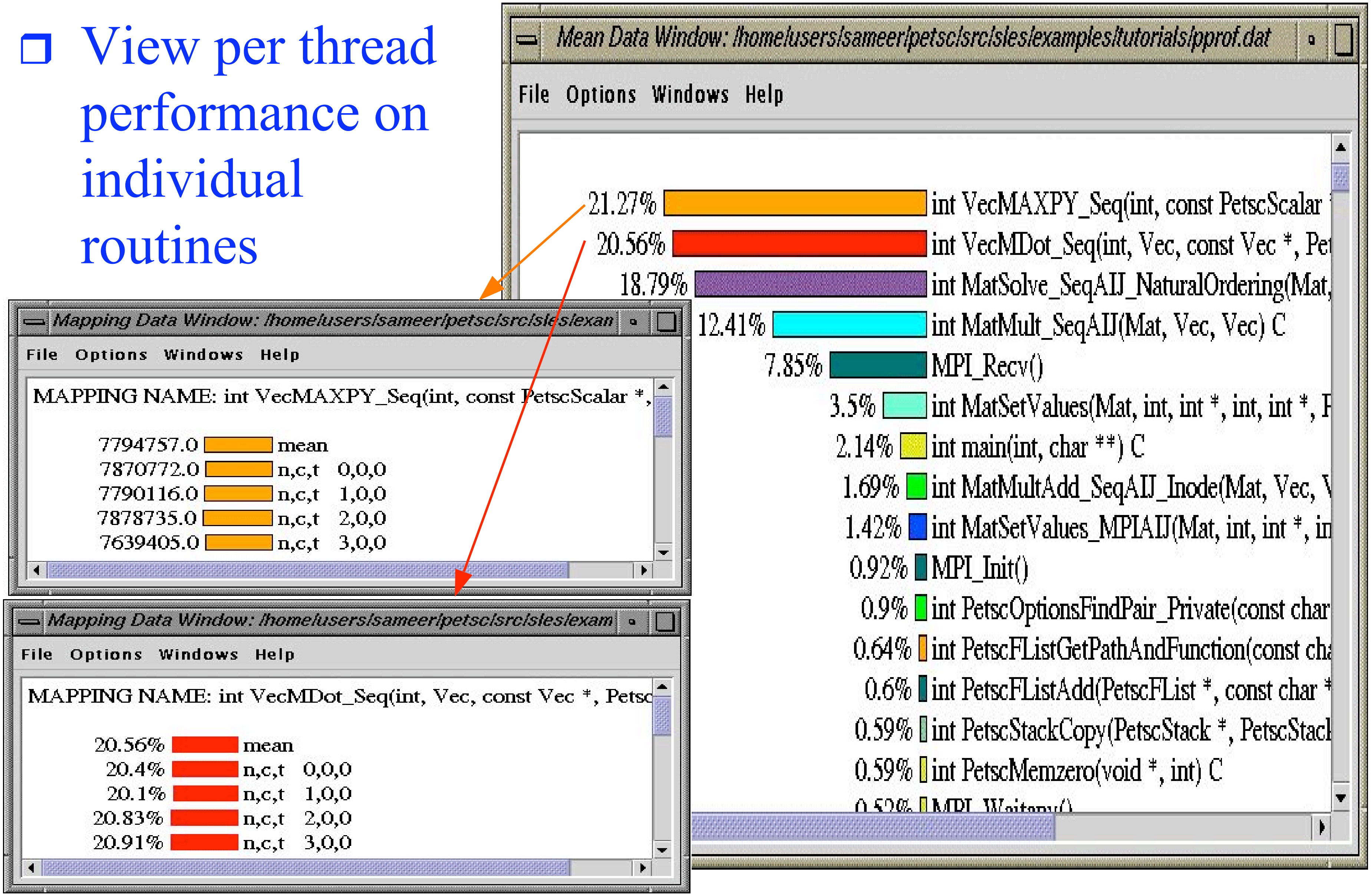
PETSc ex2(Profile - overall and message counts)



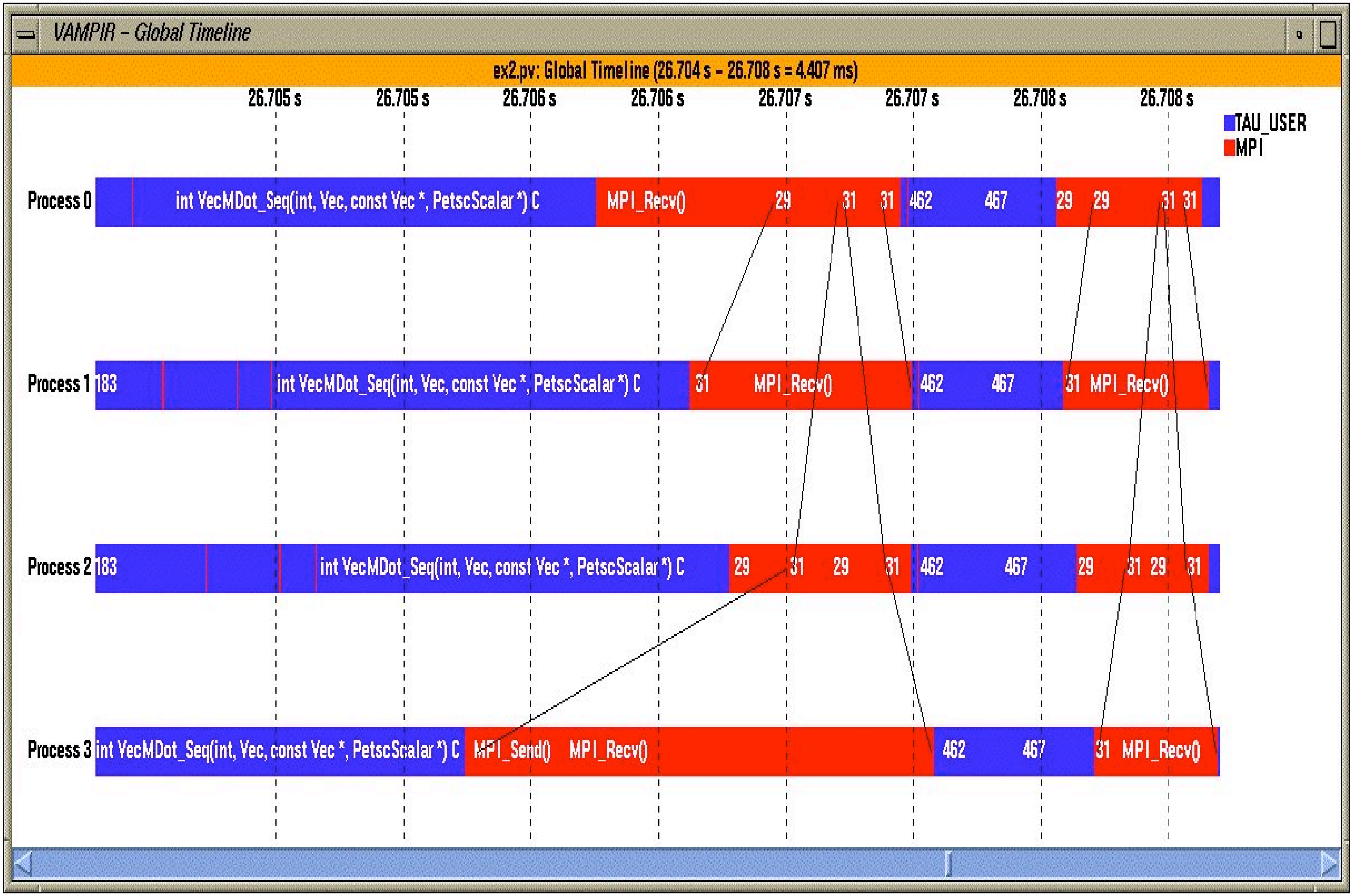
- Observe load balance
- Track messages

PETSc ex2 (Profile - percentages and time)

- View per thread performance on individual routines



PETSc ex2 (Trace)



PETSc ex19

- Non-linear solver (SNES)
  - 2-D driven cavity code
  - Uses velocity-vorticity formulation
  - Finite difference discretization on a structured grid
- Problem size and measurements
  - 56x56 mesh size on quad Pentium III (550 Mhz, Linux)
  - Executes for approximately one minute
  - MPI wrapper interposition library
  - PDT (*tau_instrumentor*)
  - Selective instrumentation (*tau_reduce*)
    - three routines identified with high instrumentation overhead

PETSc ex19 (Profile - wallclock time)

Mean Total Stat Window: /data/sameer/petsc/src/snes/examples/tutorials/data/WALLCLOCK_TIME/

File Options Windows Help

Sorted by inclusive time

%time	msec	total msec	#call	#subrs	usec/call	name
100.0	19	1:00.337	1	63	60337067	int main(int, char **)
93.7	0.117	56,558	2	10	28279316	int DMMGSolve(DMMG *)
93.7	0.0477	56,556	2	2	28278247	int DMMGSolveSNES(DMMG *, int)
93.7						
93.7						
88.0						
50.6						
50.6						
50.6						
46.4						
46.4						
46.4						
45.7						
38.3						
38.2						
37.3						
37.3						

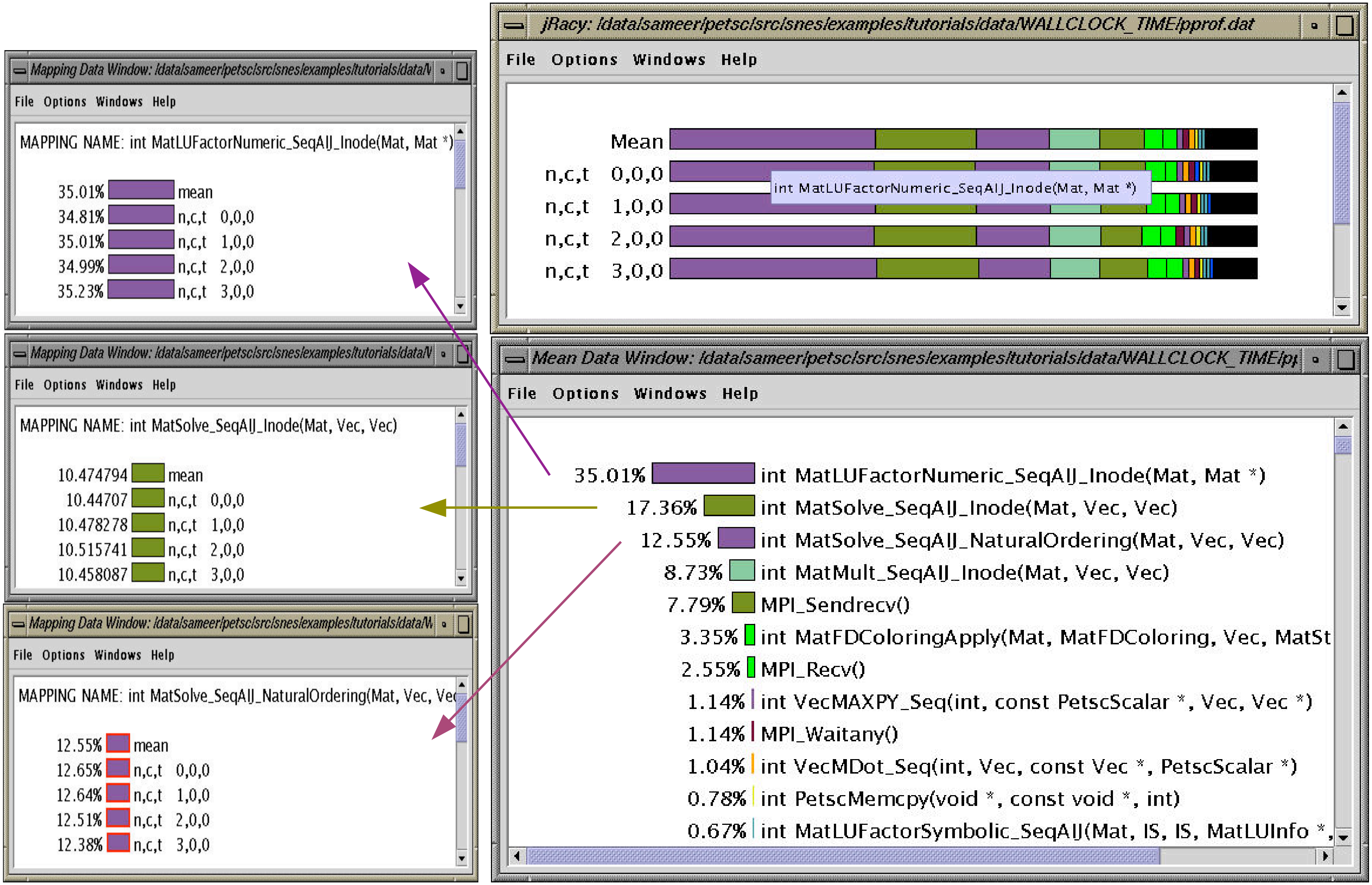
Mean Total Stat Window: /data/sameer/petsc/src/snes/examples/tutorials/data/WALLCLOCK_TIME/ppro

File Options Windows Help

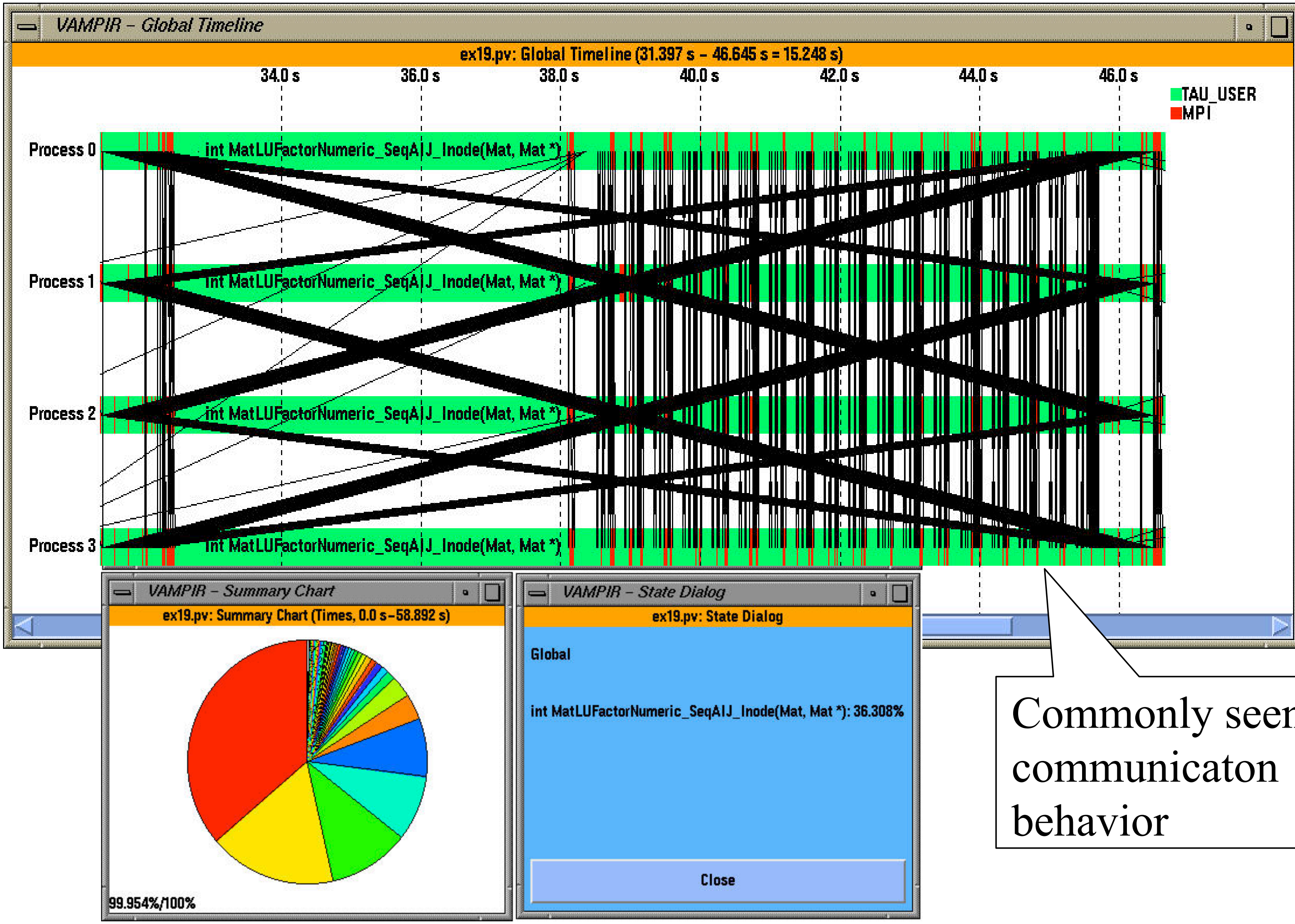
Sorted by exclusive time

%time	msec	total msec	#call	#subrs	usec/call	name
35.0	21,123	21,136	6	90	3522740	int MatLUFactorNumeric_SeqAIJ_Inode(Mat,
17.4	10,474	10,479	68	544	154111	int MatSolve_SeqAIJ_Inode(Mat, Vec, Vec)
12.6	7,570	7,574	136	544	55694	int MatSolve_SeqAIJ_NaturalOrdering(Mat,
8.7	5,267	5,272	208	832	25351	int MatMult_SeqAIJ_Inode(Mat, Vec, Vec)
7.8	4,702	4,704	1212	1212	3881	MPI_Sendrecv()
5.2	2,020	3,151	8	1280	393963	int MatFDColoringApply(Mat, MatFDColorin
2.5	1,536	1,536	994.75	0	1545	MPI_Recv()
1.1	688	688	242	0	2847	int VecMAXPY_Seq(int, const PetscScalar
1.1	686	686	978.5	0	701	MPI_Waitany()
1.0	629	629	170	0	3703	int VecMDot_Seq(int, Vec, const Vec *, P
0.8	470	470	1075	0	437	int PetscMemcpy(void *, const void *, in
1.3	404	797	2	52	398722	int MatLUFactorSymbolic_SeqAIJ(Mat, IS,
0.7	400	400	3934	0	102	int PetscMemzero(void *, int)
0.6	356	359	208	832	1726	int MatMultAdd_SeqAIJ_Inode(Mat, Vec, Ve
0.5	291	291	1	35	291261	MPI_Init()
0.7	253	414	386	4632	1074	int VecScatterBegin_PtoP(Vec, Vec, Inser
0.4	252	253	48	82	5284	int Mat_AIJ_CheckInode(Mat, PetscTruth)

PETSc ex19 (Profile - overall and percentages)

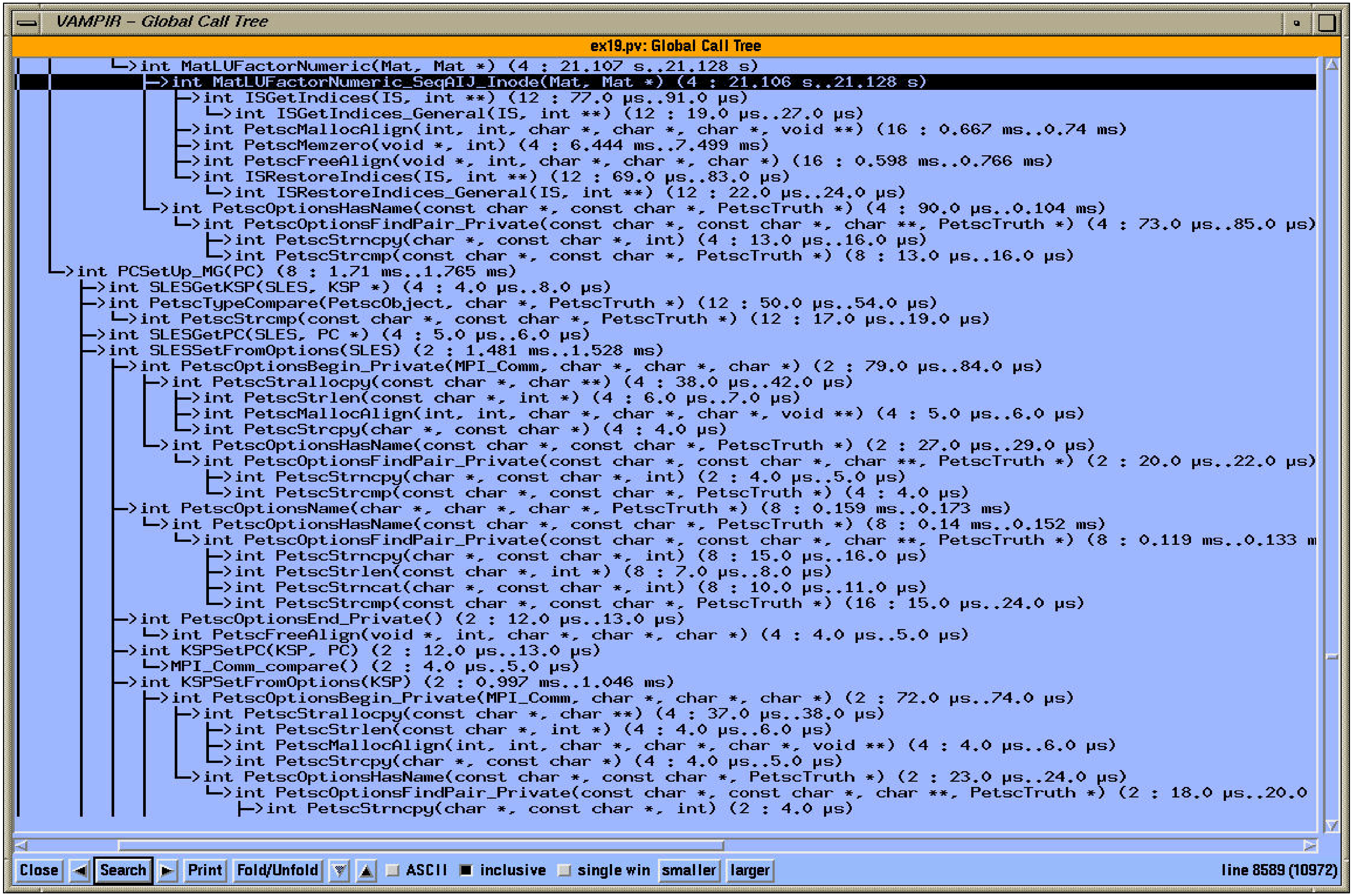


PETSc ex19 (Tracing)



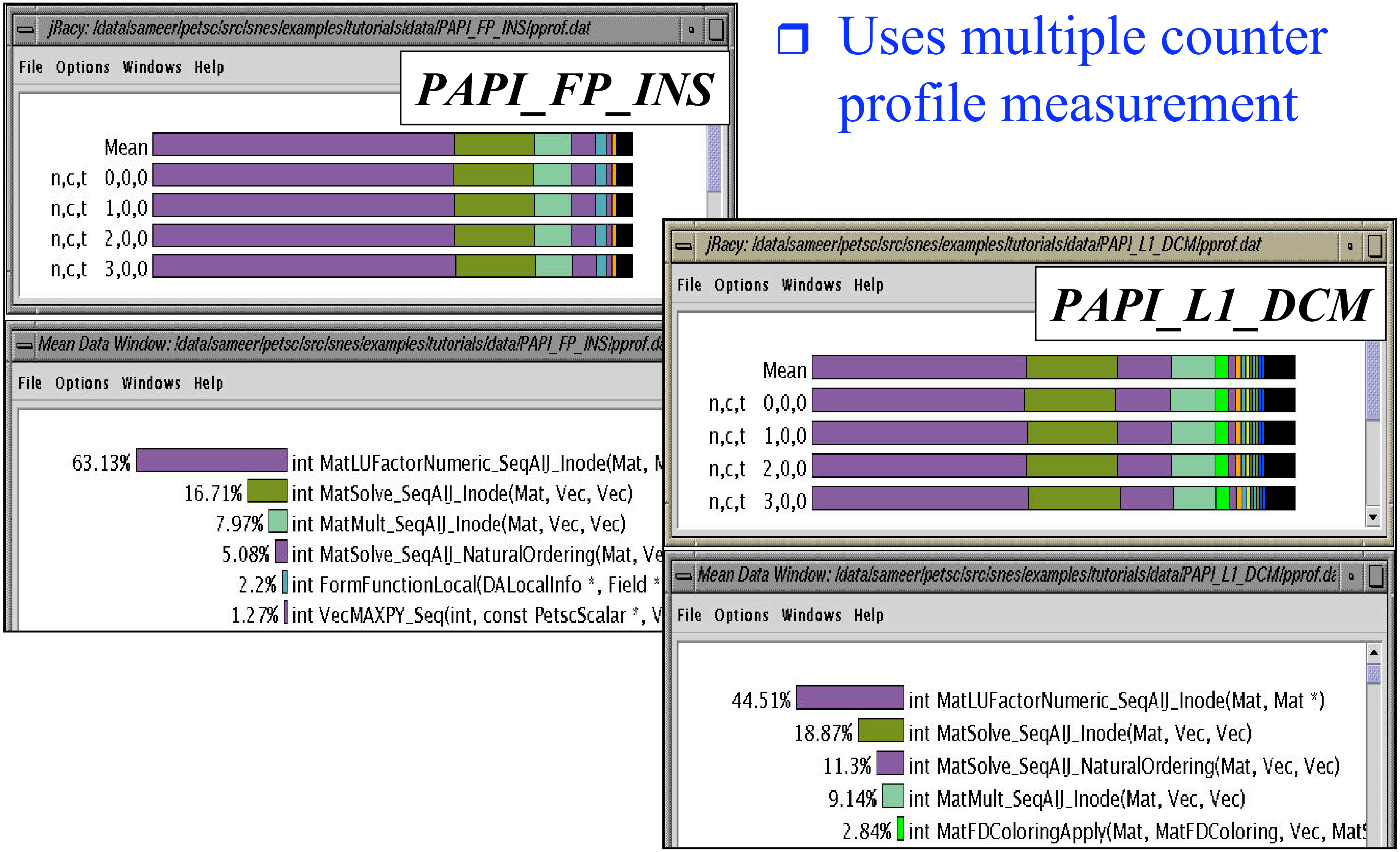
Commonly seen communicaton behavior

PETSc ex19 (Tracing - callgraph)




```
└─>int MatLUFactorNumeric(Mat, Mat *) (4 : 21.107 s..21.128 s)
   └─>int MatLUFactorNumeric_SeqAIJ_Inode(Mat, Mat *) (4 : 21.106 s..21.128 s)
      ├─>int ISGetIndices(IS, int **) (12 : 77.0 µs..91.0 µs)
      │  └─>int ISGetIndices_General(IS, int **) (12 : 19.0 µs..27.0 µs)
      ├─>int PetscMallocAlign(int, int, char *, char *, char *, void **) (16 : 0.667 ms..0.74 ms)
      ├─>int PetscMemzero(void *, int) (4 : 6.444 ms..7.499 ms)
      ├─>int PetscFreeAlign(void *, int, char *, char *, char *) (16 : 0.598 ms..0.766 ms)
      └─>int ISRestoreIndices(IS, int **) (12 : 69.0 µs..83.0 µs)
         └─>int ISRestoreIndices_General(IS, int **) (12 : 22.0 µs..24.0 µs)
   └─>int PetscOptionsHasName(const char *, const char *, PetscTruth *) (4 : 90.0 µs..0.104 ms)
      └─>int PetscOptionsFindPair_Private(const char *, const char *, char **, PetscTruth *) (4 : 73.0 µs..85.0 µs)
         ├─>int PetscStrncpy(char *, const char *, int) (4 : 13.0 µs..16.0 µs)
         └─>int PetscStrcmp(const char *, const char *, PetscTruth *) (8 : 13.0 µs..16.0 µs)
└─>int PCSetUp_MG(PC) (8 : 1.71 ms..1.765 ms)
   ├─>int SLESGetKSP(SLES, KSP *) (4 : 4.0 µs..8.0 µs)
   ├─>int PetscTypeCompare(PetscObject, char *, PetscTruth *) (12 : 50.0 µs..54.0 µs)
   │  └─>int PetscStrcmp(const char *, const char *, PetscTruth *) (12 : 17.0 µs..19.0 µs)
   ├─>int SLESGetPC(SLES, PC *) (4 : 5.0 µs..6.0 µs)
   ├─>int SLESSetFromOptions(SLES) (2 : 1.481 ms..1.528 ms)
   │  ├─>int PetscOptionsBegin_Private(MPI_Comm, char *, char *, char *) (2 : 79.0 µs..84.0 µs)
   │  │  ├─>int PetscStrallocpy(const char *, char **) (4 : 38.0 µs..42.0 µs)
   │  │  │  ├─>int PetscStrlen(const char *, int *) (4 : 6.0 µs..7.0 µs)
   │  │  │  ├─>int PetscMallocAlign(int, int, char *, char *, char *, void **) (4 : 5.0 µs..6.0 µs)
   │  │  │  └─>int PetscStrcpy(char *, const char *) (4 : 4.0 µs)
   │  │  └─>int PetscOptionsHasName(const char *, const char *, PetscTruth *) (2 : 27.0 µs..29.0 µs)
   │  │     └─>int PetscOptionsFindPair_Private(const char *, const char *, char **, PetscTruth *) (2 : 20.0 µs..22.0 µs)
   │  │        ├─>int PetscStrncpy(char *, const char *, int) (2 : 4.0 µs..5.0 µs)
   │  │        └─>int PetscStrcmp(const char *, const char *, PetscTruth *) (4 : 4.0 µs)
   │  ├─>int PetscOptionsName(char *, char *, char *, PetscTruth *) (8 : 0.159 ms..0.173 ms)
   │  │  └─>int PetscOptionsHasName(const char *, const char *, PetscTruth *) (8 : 0.14 ms..0.152 ms)
   │  │     └─>int PetscOptionsFindPair_Private(const char *, const char *, char **, PetscTruth *) (8 : 0.119 ms..0.133 m
   │  │        ├─>int PetscStrncpy(char *, const char *, int) (8 : 15.0 µs..16.0 µs)
   │  │        ├─>int PetscStrlen(const char *, int *) (8 : 7.0 µs..8.0 µs)
   │  │        ├─>int PetscStrncat(char *, const char *, int) (8 : 10.0 µs..11.0 µs)
   │  │        └─>int PetscStrcmp(const char *, const char *, PetscTruth *) (16 : 15.0 µs..24.0 µs)
   │  ├─>int PetscOptionsEnd_Private() (2 : 12.0 µs..13.0 µs)
   │  │  └─>int PetscFreeAlign(void *, int, char *, char *, char *) (4 : 4.0 µs..5.0 µs)
   │  ├─>int KSPSetPC(KSP, PC) (2 : 12.0 µs..13.0 µs)
   │  │  └─>MPI_Comm_compare() (2 : 4.0 µs..5.0 µs)
   │  ├─>int KSPSetFromOptions(KSP) (2 : 0.997 ms..1.046 ms)
   │  │  ├─>int PetscOptionsBegin_Private(MPI_Comm, char *, char *, char *) (2 : 72.0 µs..74.0 µs)
   │  │  │  ├─>int PetscStrallocpy(const char *, char **) (4 : 37.0 µs..38.0 µs)
   │  │  │  │  ├─>int PetscStrlen(const char *, int *) (4 : 4.0 µs..6.0 µs)
   │  │  │  │  ├─>int PetscMallocAlign(int, int, char *, char *, char *, void **) (4 : 4.0 µs..6.0 µs)
   │  │  │  │  └─>int PetscStrcpy(char *, const char *) (4 : 4.0 µs..5.0 µs)
   │  │  │  └─>int PetscOptionsHasName(const char *, const char *, PetscTruth *) (2 : 23.0 µs..24.0 µs)
   │  │  │     └─>int PetscOptionsFindPair_Private(const char *, const char *, char **, PetscTruth *) (2 : 18.0 µs..20.0
   │  │  │        ├─>int PetscStrncpy(char *, const char *, int) (2 : 4.0 µs)
```

PETSc ex19 (PAPI_FP_INS, PAPI_L1_DCM)



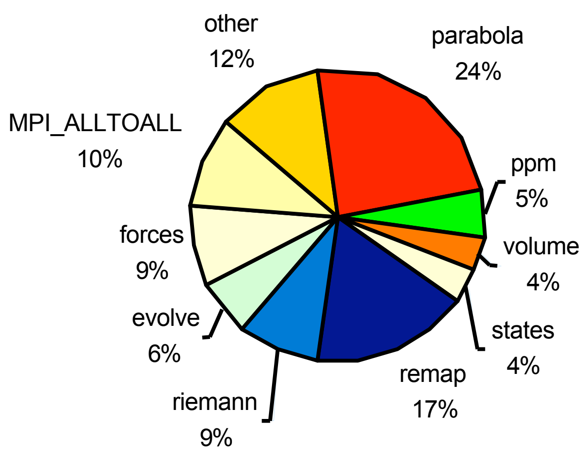
- Uses multiple counter profile measurement

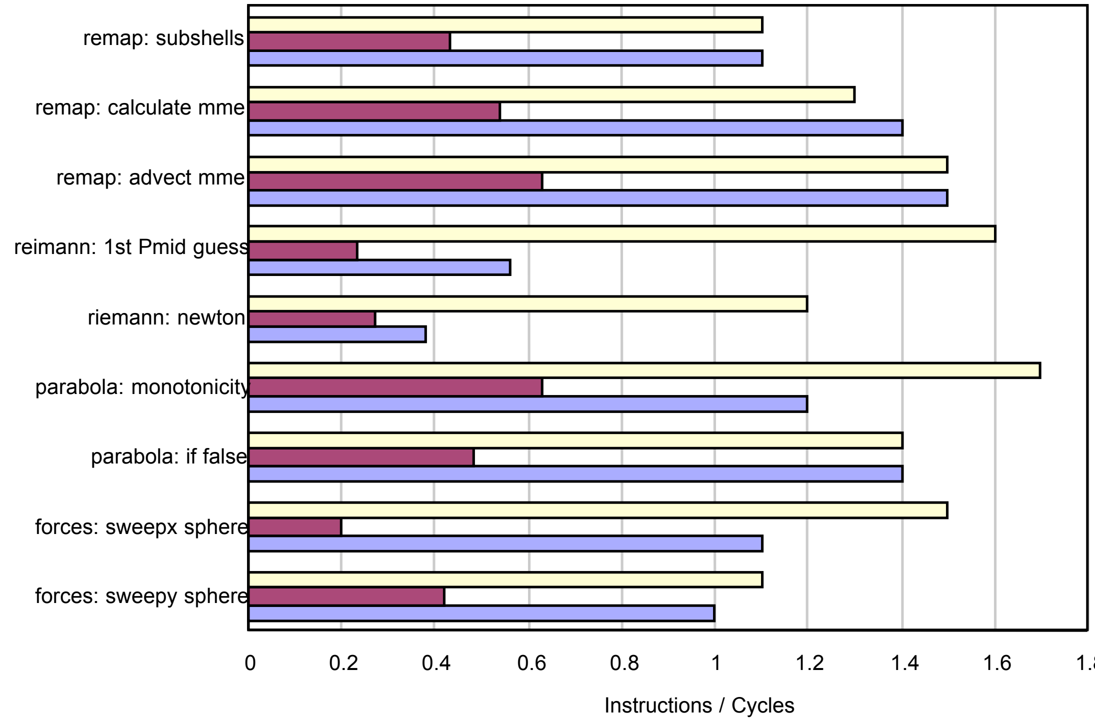
EVH1 – High Energy and Nuclear Physics

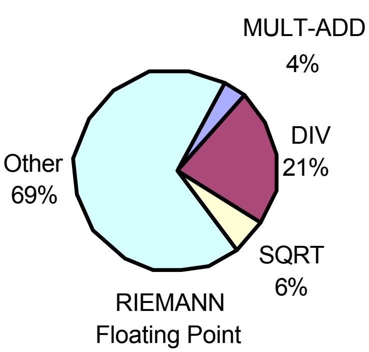
- Enhanced Virginia Hydrodynamics #1 (EVH1)
  - "TeraScale Simulations of Neutrino-Driven Supernovae and Their Nucleosynthesis" SciDAC project
  - Configured to run a simulation of the Sedov-Taylor blast wave solution in 2D spherical geometry
- EVH1 communication bound for > 64 processors
  - Predominant routine (>50% of execution time) at this scale is MPI_ALLTOALL
  - Used in matrix transpose-like operations
  - Current implementation uses 1D matrix decomposition
- PERC benchmark code

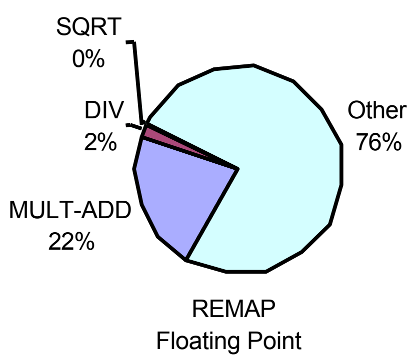
EVH1 Aggregate Performance

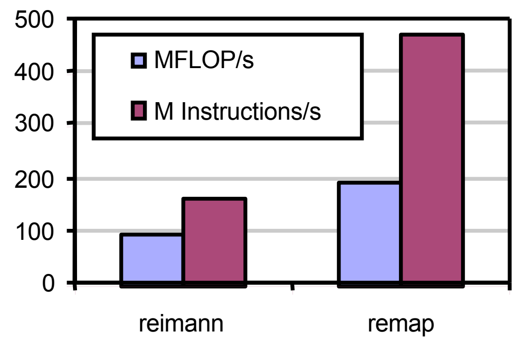
- Aggregate performance measures over all tasks for .1 second simulation
  - Using PAPI on IBM SP



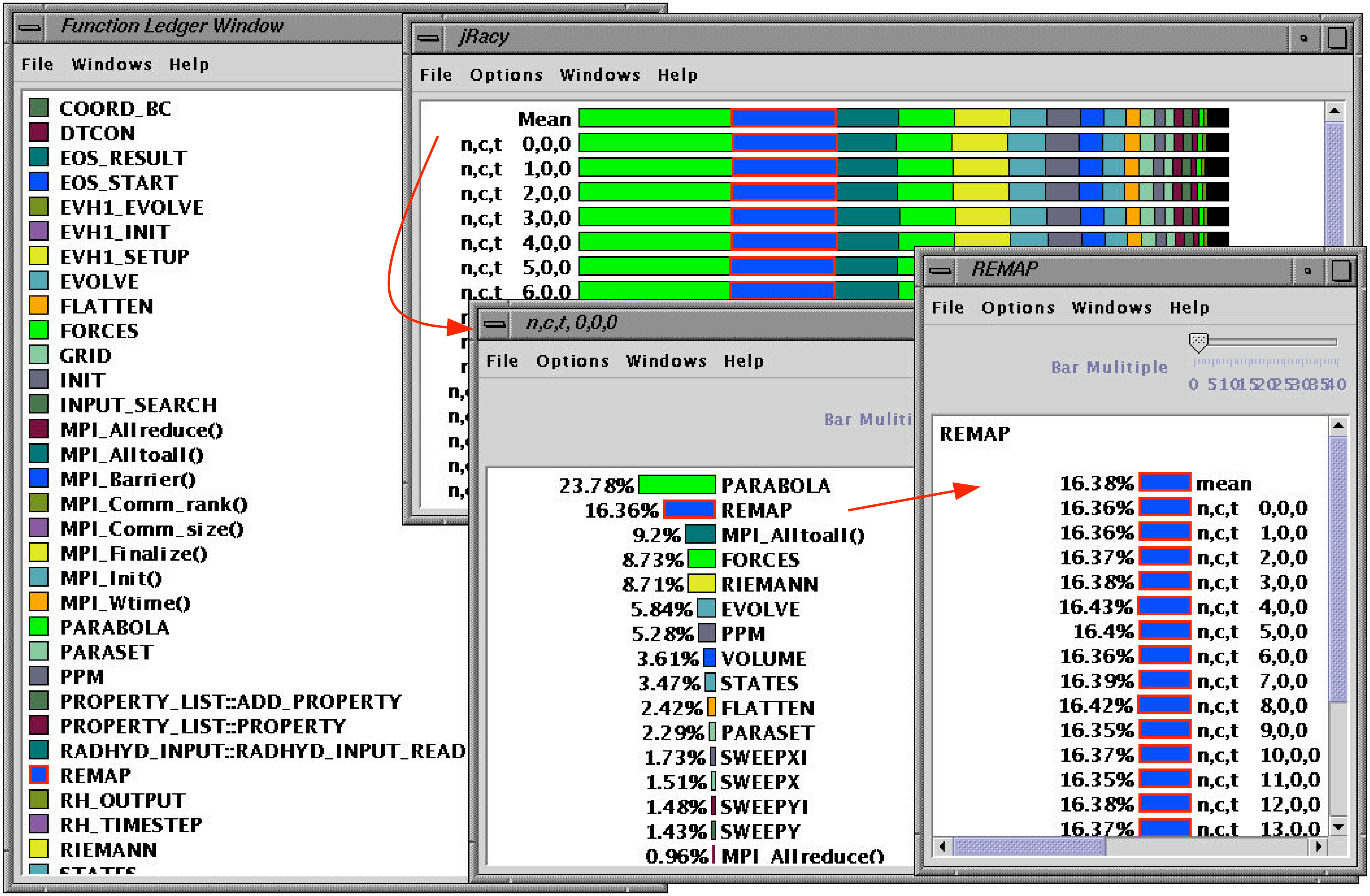




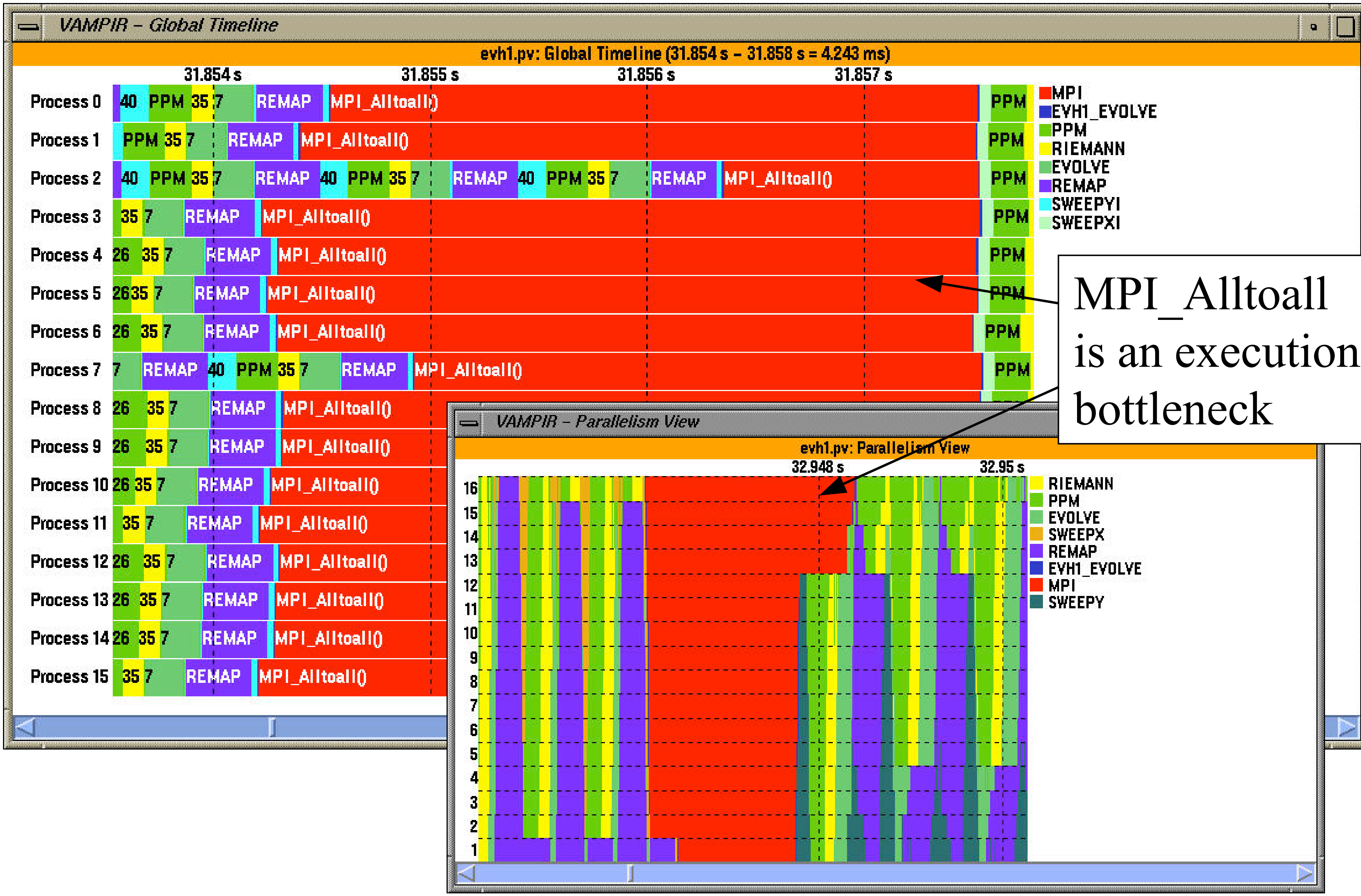




EVH1 Execution Profile



EVH1 Execution Trace



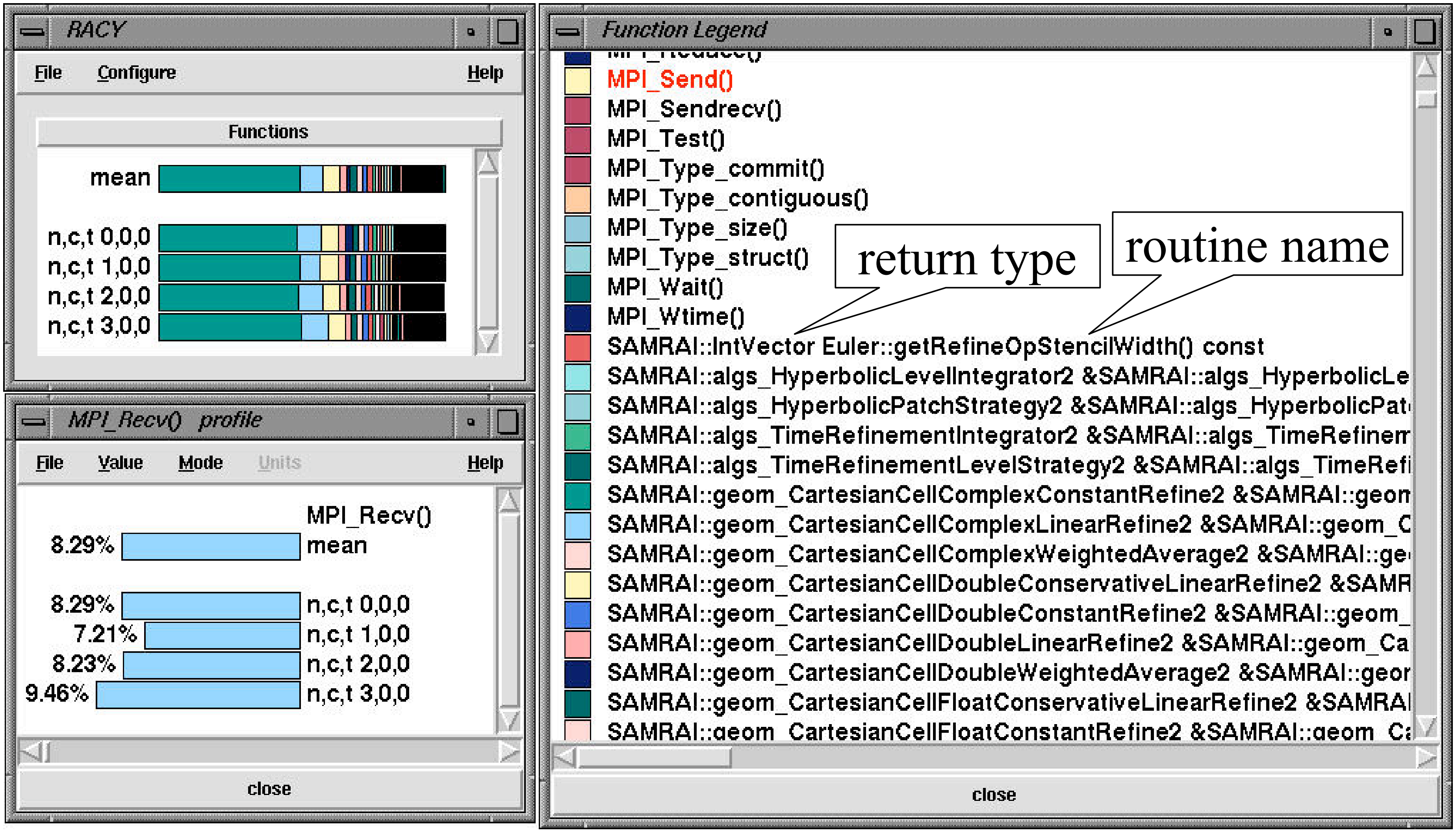
MPI_Alltoall is an execution bottleneck

SAMRAI

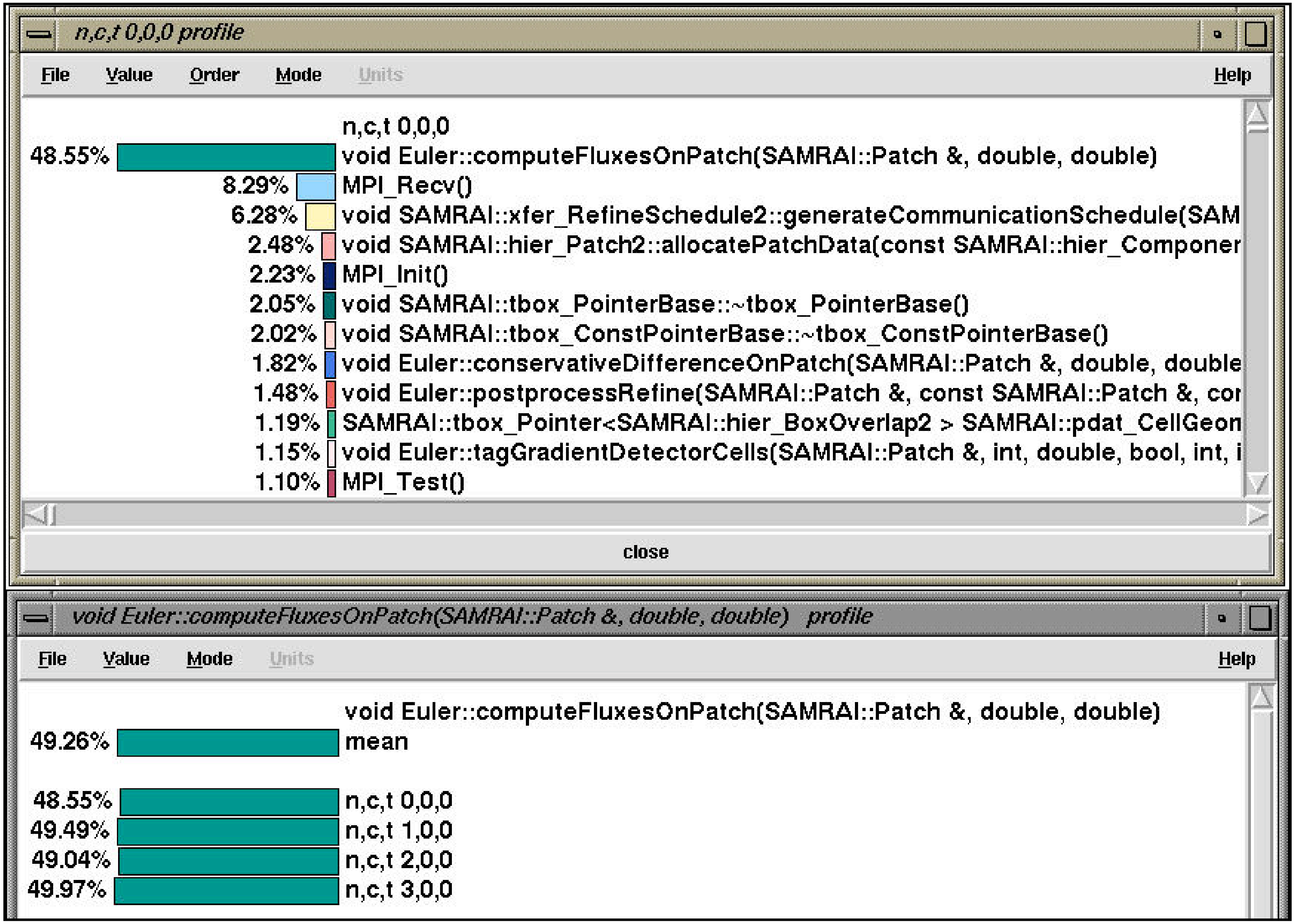
- Structured Adaptive Mesh Refinement Application Infrastructure (SAMRAI)
  - Andy Wissink (LLNL)
- Programming
  - C++ and MPI
  - SPMD
- Instrumentation
  - PDT for automatic instrumentation of routines
  - MPI interposition wrappers
  - SAMRAI timers for interesting code segments
    - timers classified in groups (*apps*, *mesh*, …)
    - timer groups are managed by TAU groups

SAMRAI Execution Profile

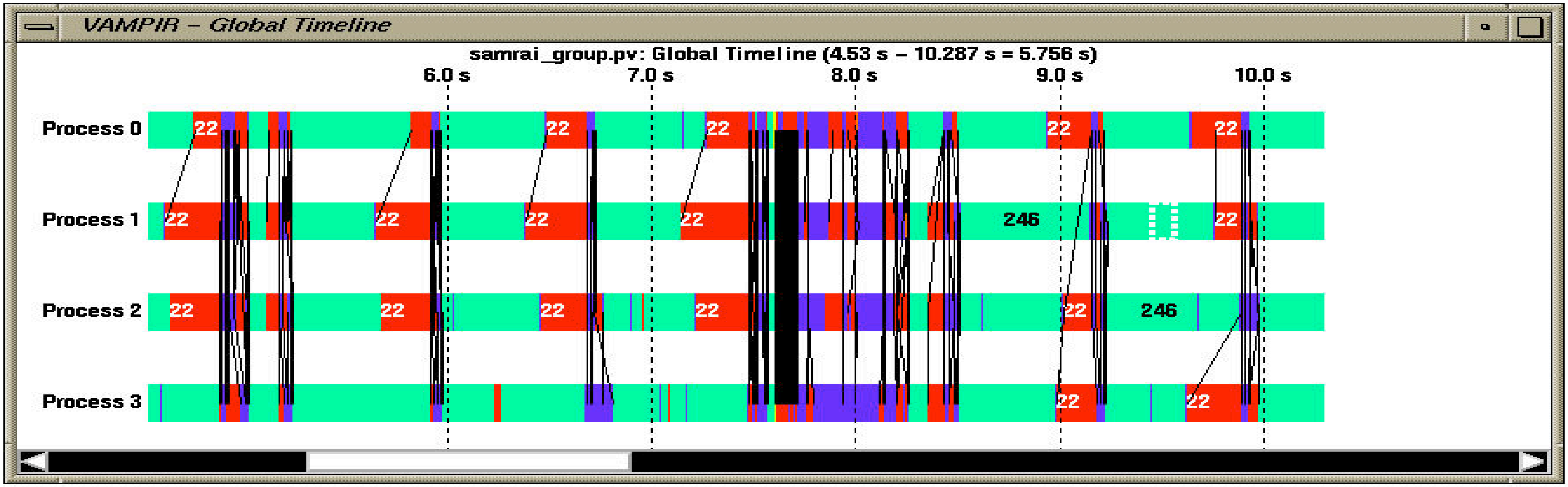
- Euler (2D)

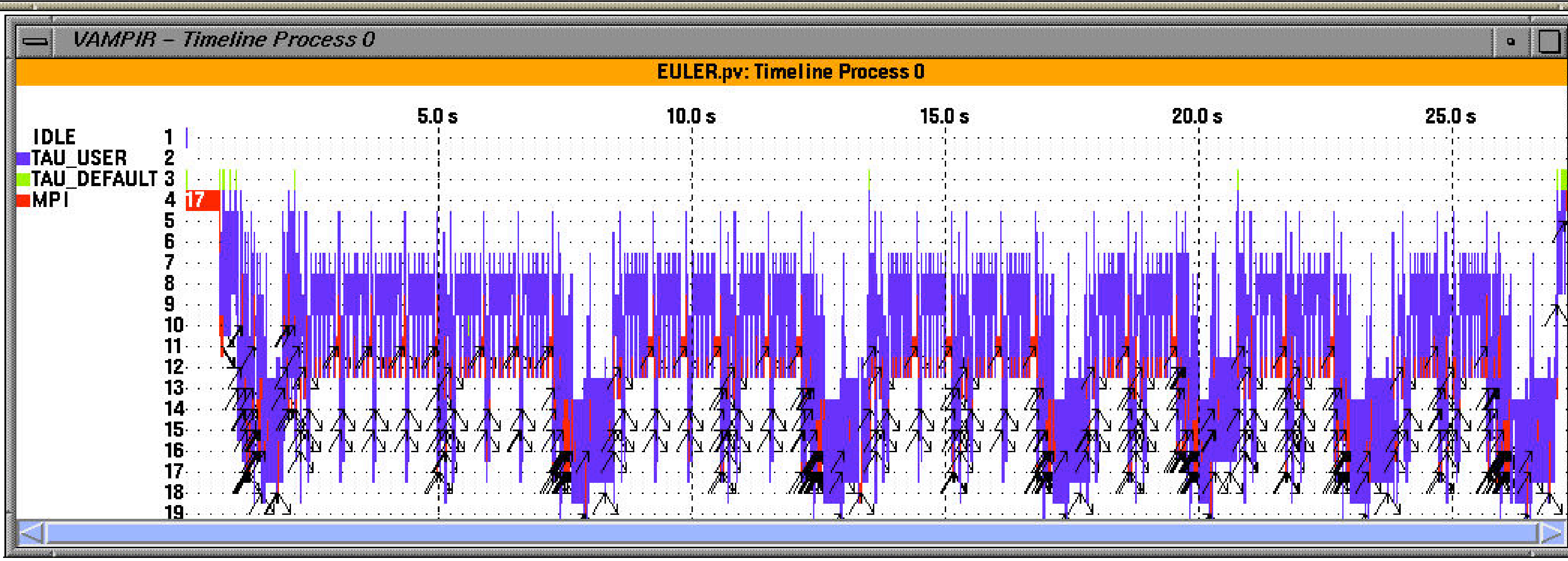


SAMRAI Euler Profile

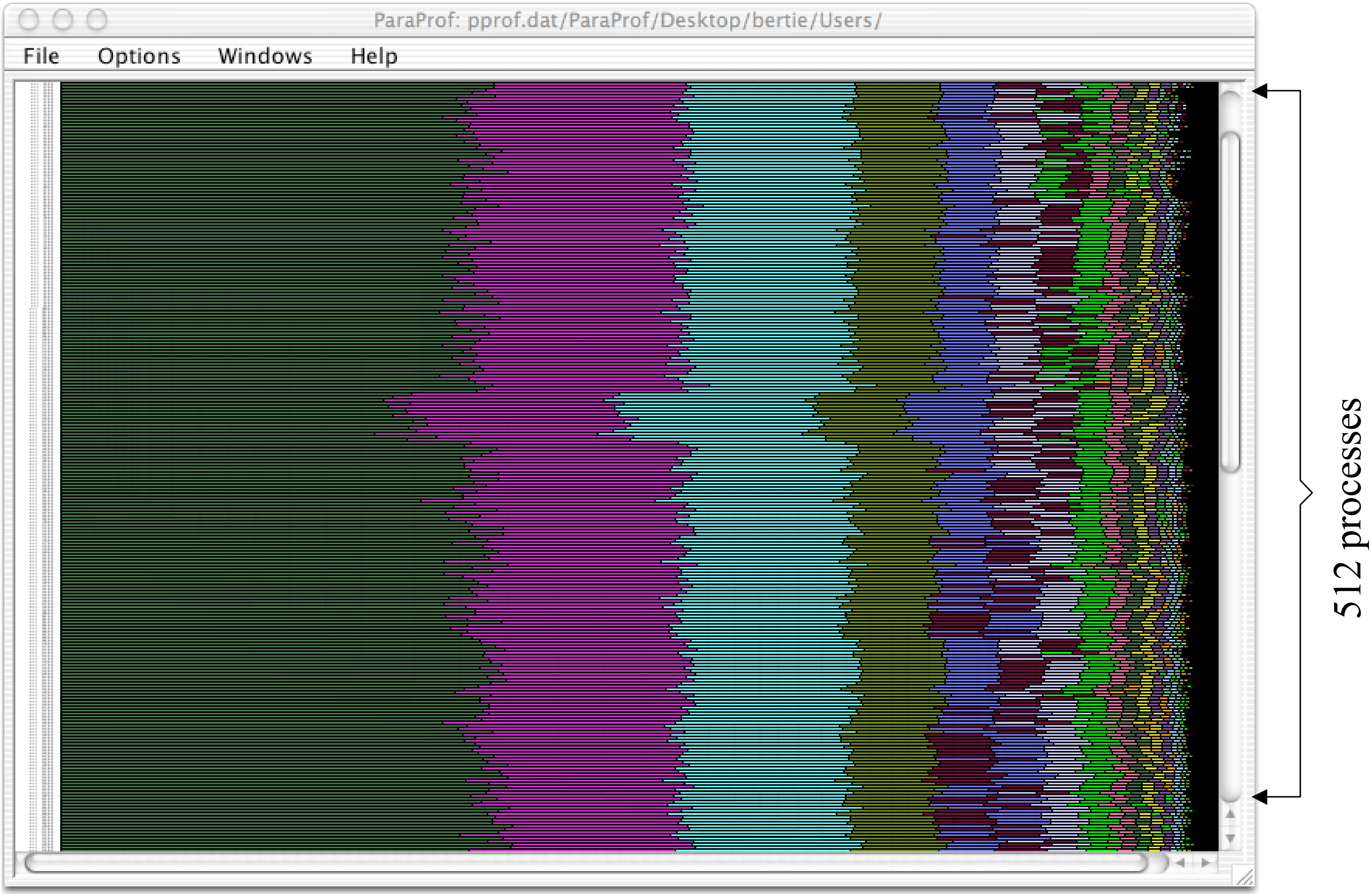


SAMRAI Euler Trace

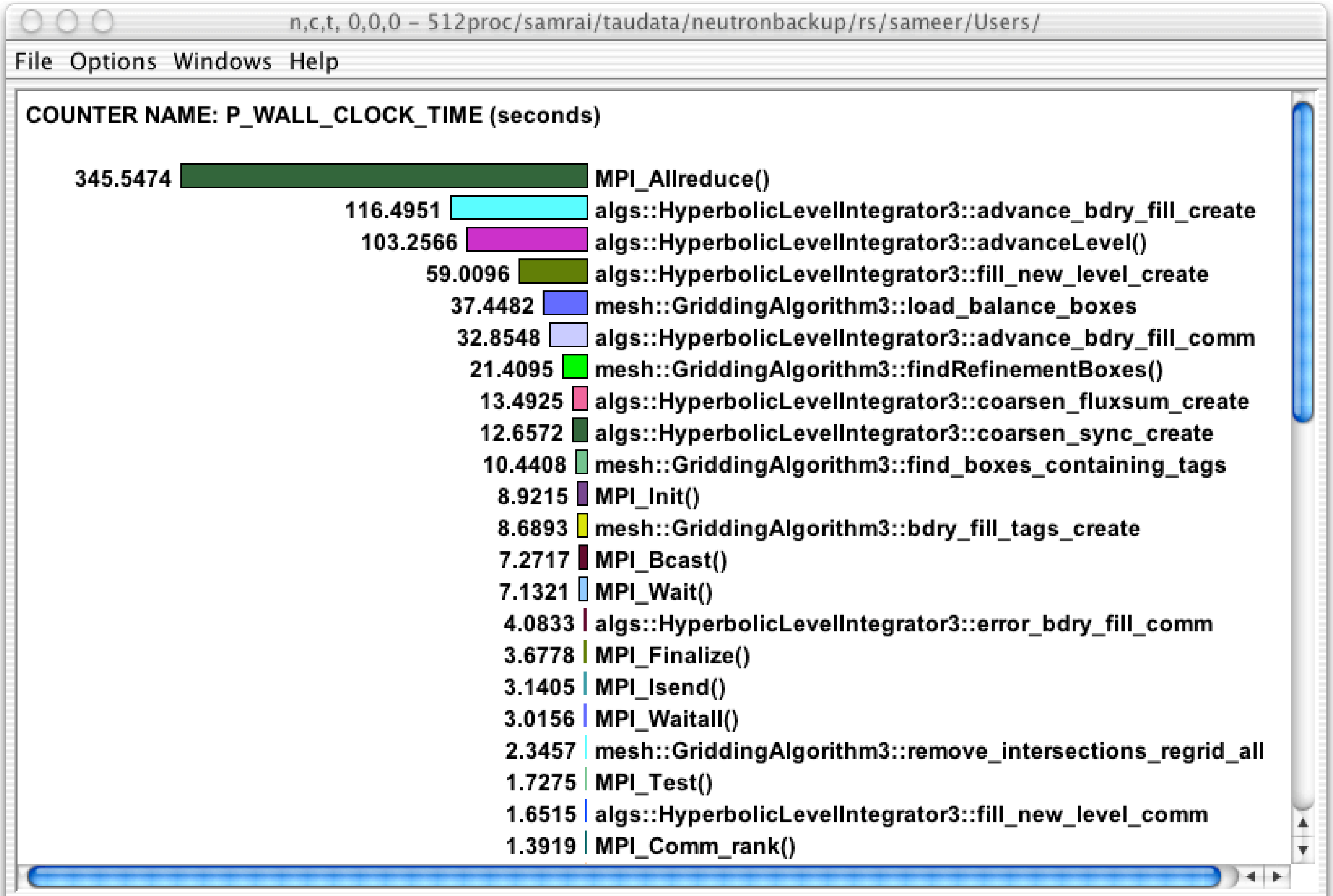




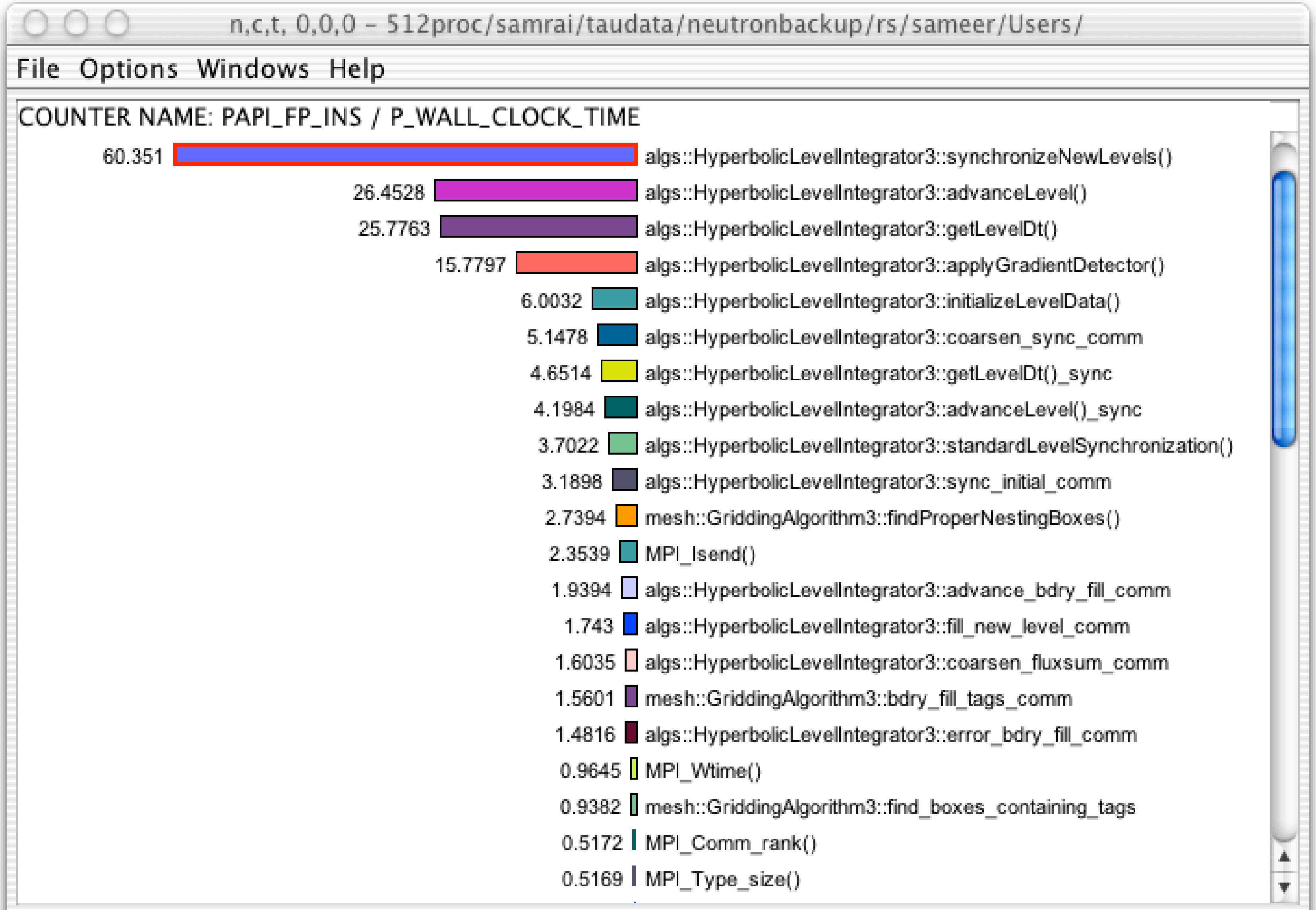
Full Profile Window (512 Processors)



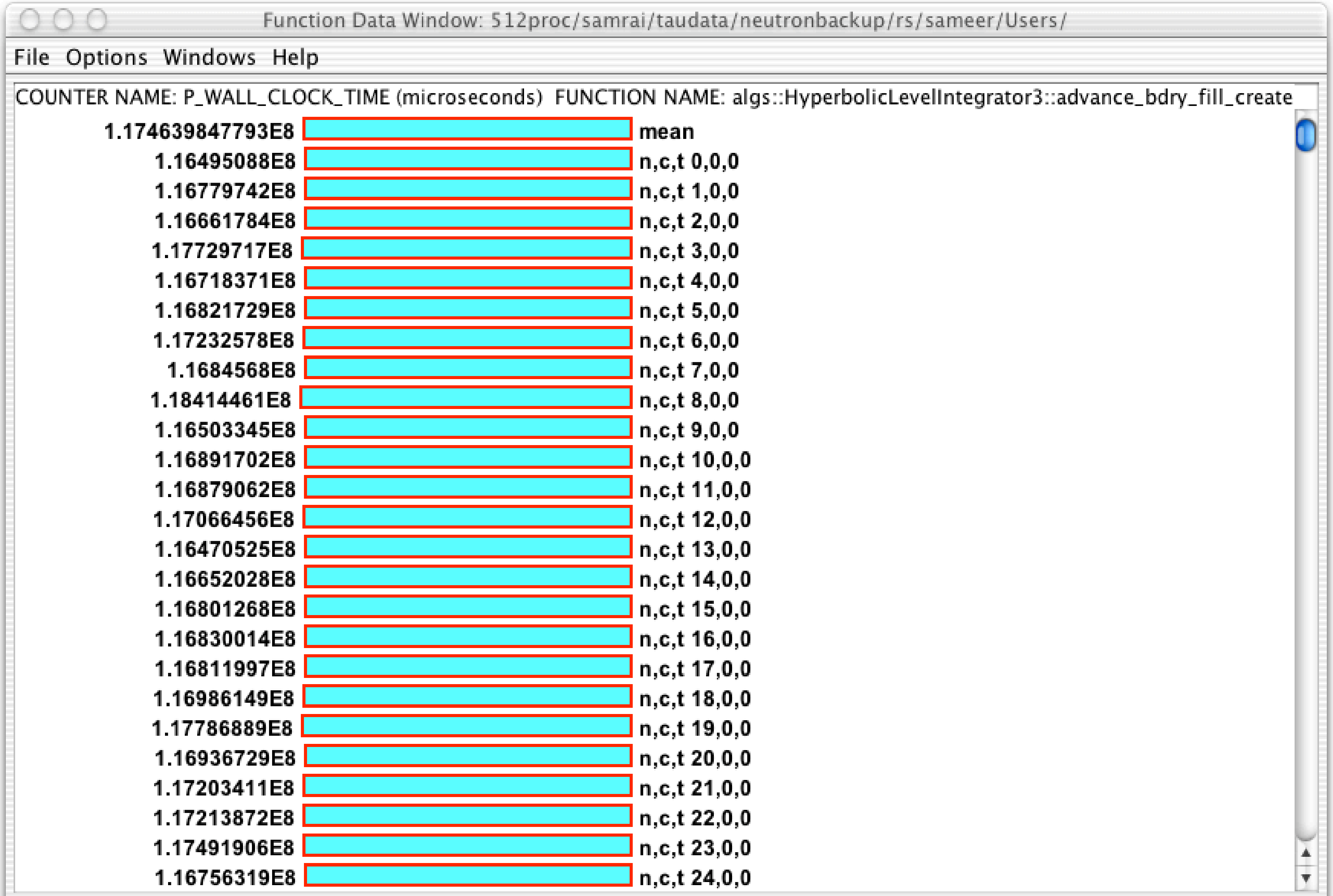
Node / Context / Thread Profile Window



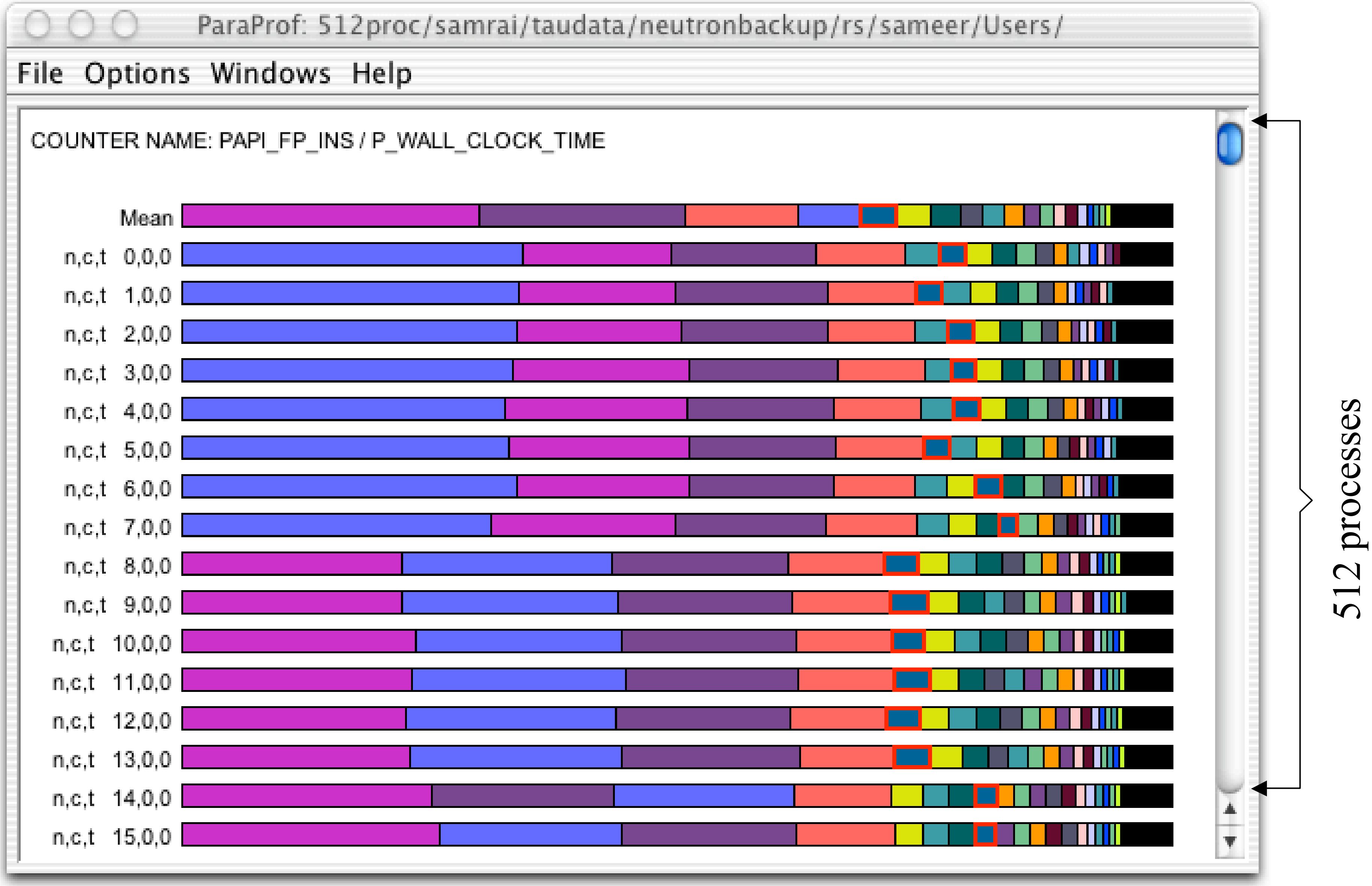
Derived Metrics



Paraprof Profile Browser Routine Window



Full Profile Window (Metric-specific)



Mixed-mode Parallel Programs (OpenMPI + MPI)

- Portable mixed-mode parallel programming
  - Multi-threaded shared memory programming
  - Inter-node message passing
- Performance measurement
  - Access to RTS and communication events
  - Associate communication and application events
- 2D Stommel model of ocean circulation
  - OpenMP for shared memory parallel programming
  - MPI for cross-box message-based parallelism
  - Jacobi iteration, 5-point stencil
  - Timothy Kaiser (San Diego Supercomputing Center)

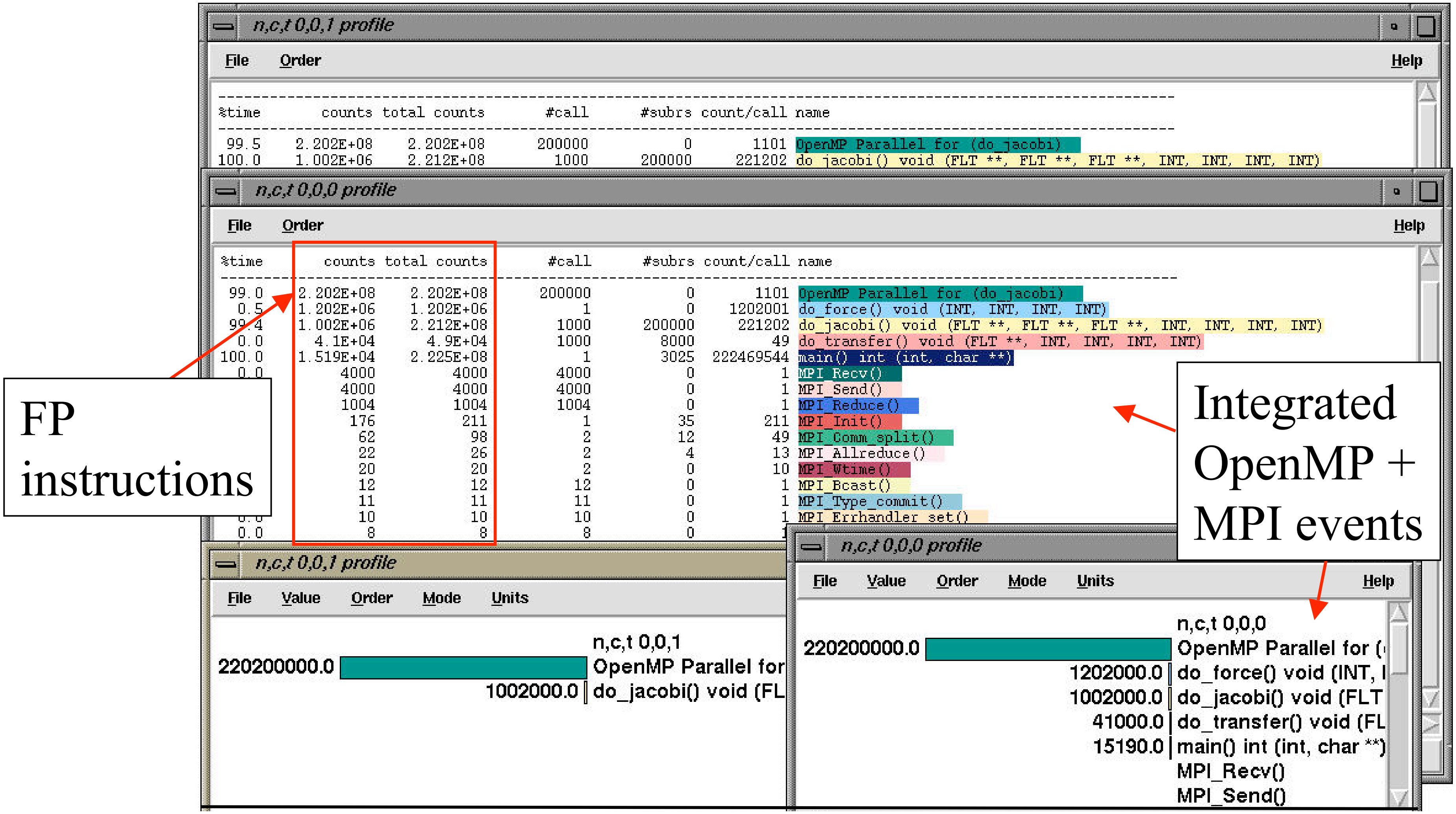
Stommel Instrumentation

- OpenMP directive instrumentation

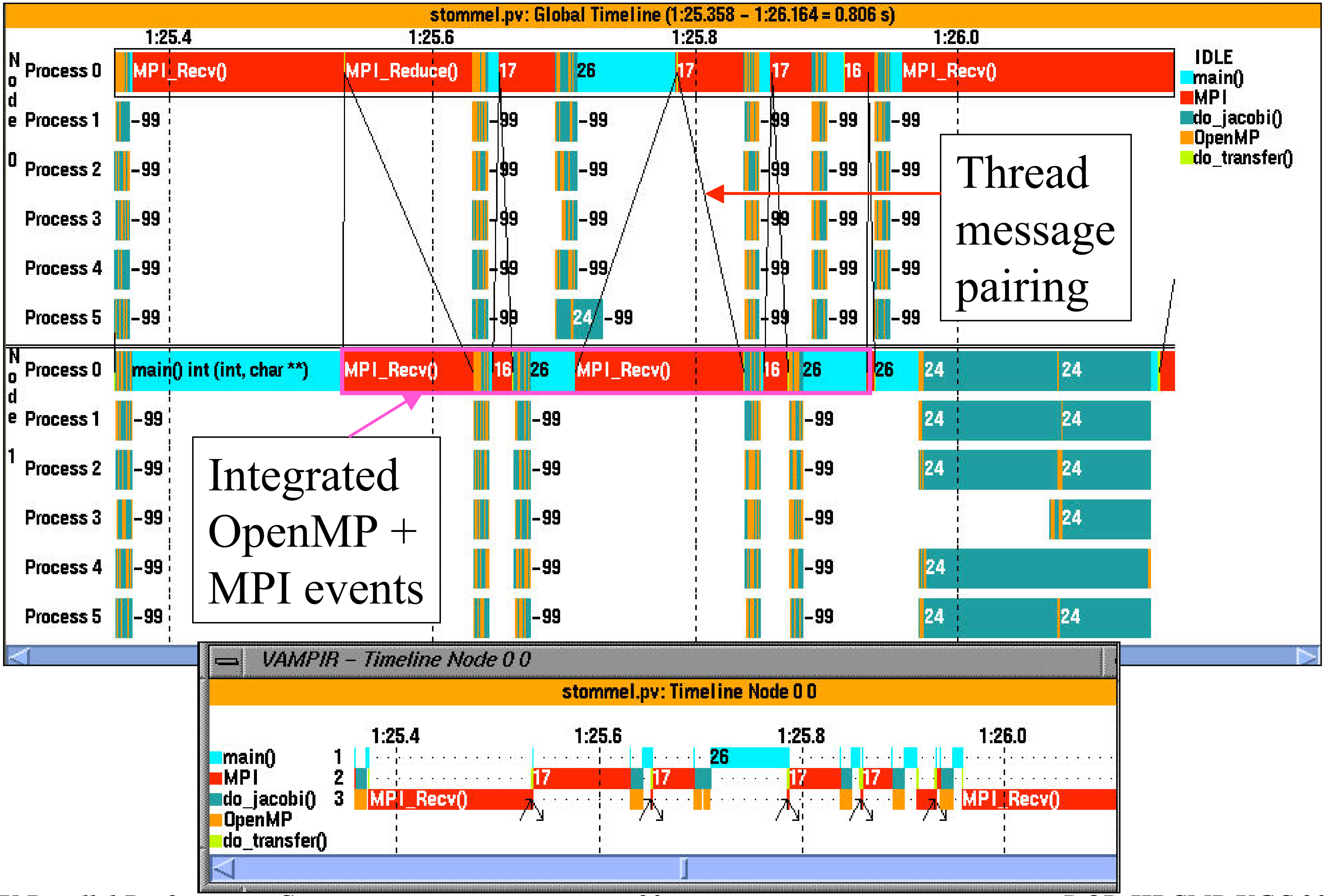
```
pomp_for_enter(&omp_rd_2);
#line 252 "stommel.c"
#pragma omp for schedule(static) reduction(+: diff) private(j)
  firstprivate (a1,a2,a3,a4,a5) nowait
for( i=i1;i<=i2;i++) {
  for(j=j1;j<=j2;j++){
    new_psi[i][j]=a1*psi[i+1][j] + a2*psi[i-1][j] + a3*psi[i][j+1]
      + a4*psi[i][j-1] - a5*the_for[i][j];
    diff=diff+fabs(new_psi[i][j]-psi[i][j]);
  }
}
pomp_barrier_enter(&omp_rd_2);
#pragma omp barrier
pomp_barrier_exit(&omp_rd_2);
pomp_for_exit(&omp_rd_2);
#line 261 "stommel.c"
```

OpenMP + MPI Ocean Modeling (HW Profile)

% configure -papi=../packages/papi -openmp -c++=pgCC -cc=pgcc -mpiinc=../packages/mpich/include -mpilib=../packages/mpich/libo



OpenMP + MPI Ocean Modeling (Trace)

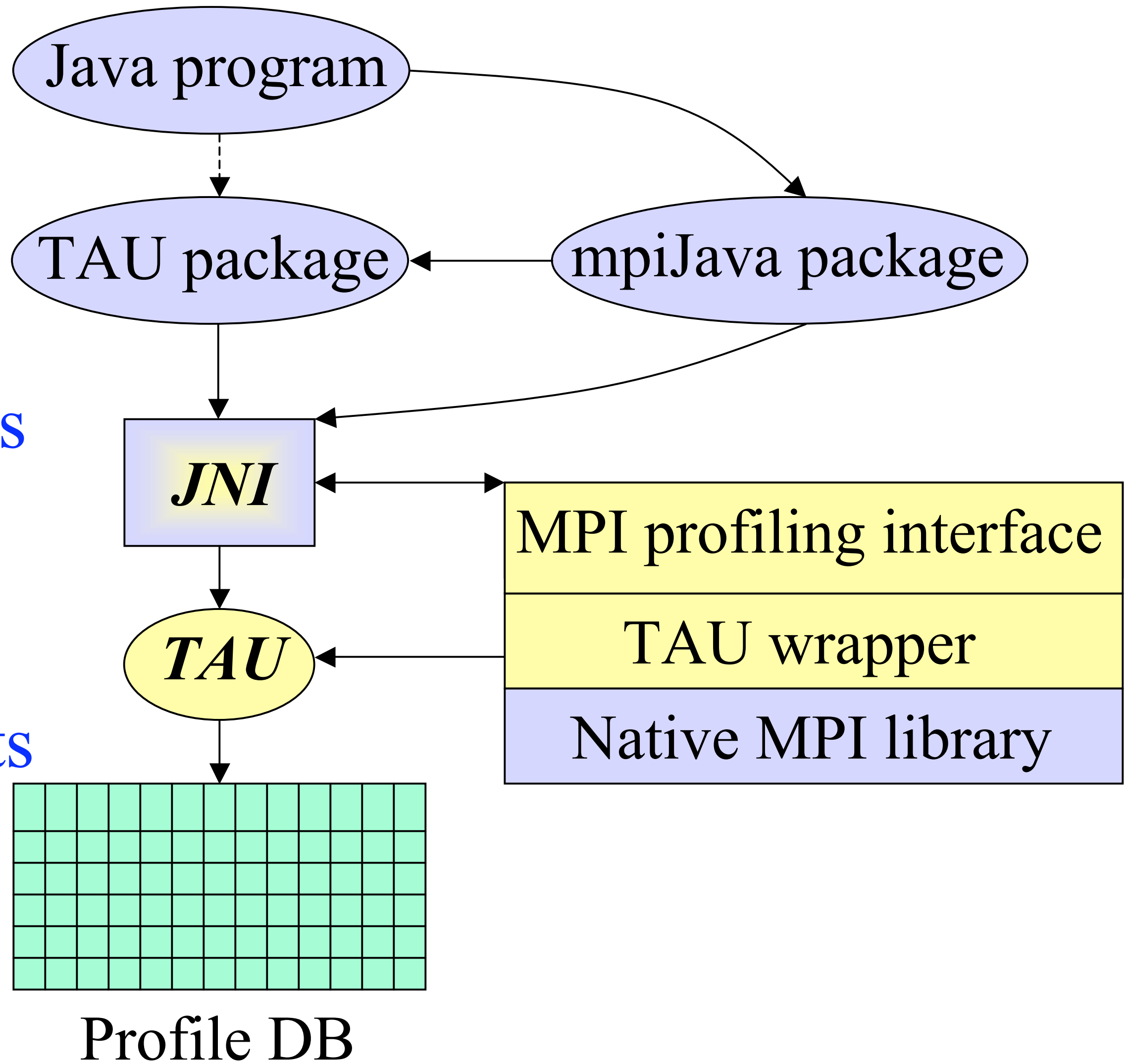


Mixed-mode Parallel Programs (Java + MPI)

- Explicit message communication libraries for Java
- MPI performance measurement
  - MPI profiling interface - link-time interposition library
  - TAU wrappers in native profiling interface library
  - Send/Receive events and communication statistics
- mpiJava (Syracuse, JavaGrande, 1999)
  - Java wrapper package
  - JNI C bindings to MPI communication library
  - Dynamic shared object (*libmpijava.so*) loaded in JVM
  - *prunjava* calls *mpirun* to distribute program to nodes
  - Contrast to Java RMI-based schemes (MPJ, CCJ)

TAU mpiJava Instrumentation Architecture

- No source instrumentation required
- Portability
- Measurement options
- Limitations
  - MPI events only
  - No mpiJava events
  - Node info only
  - No thread info



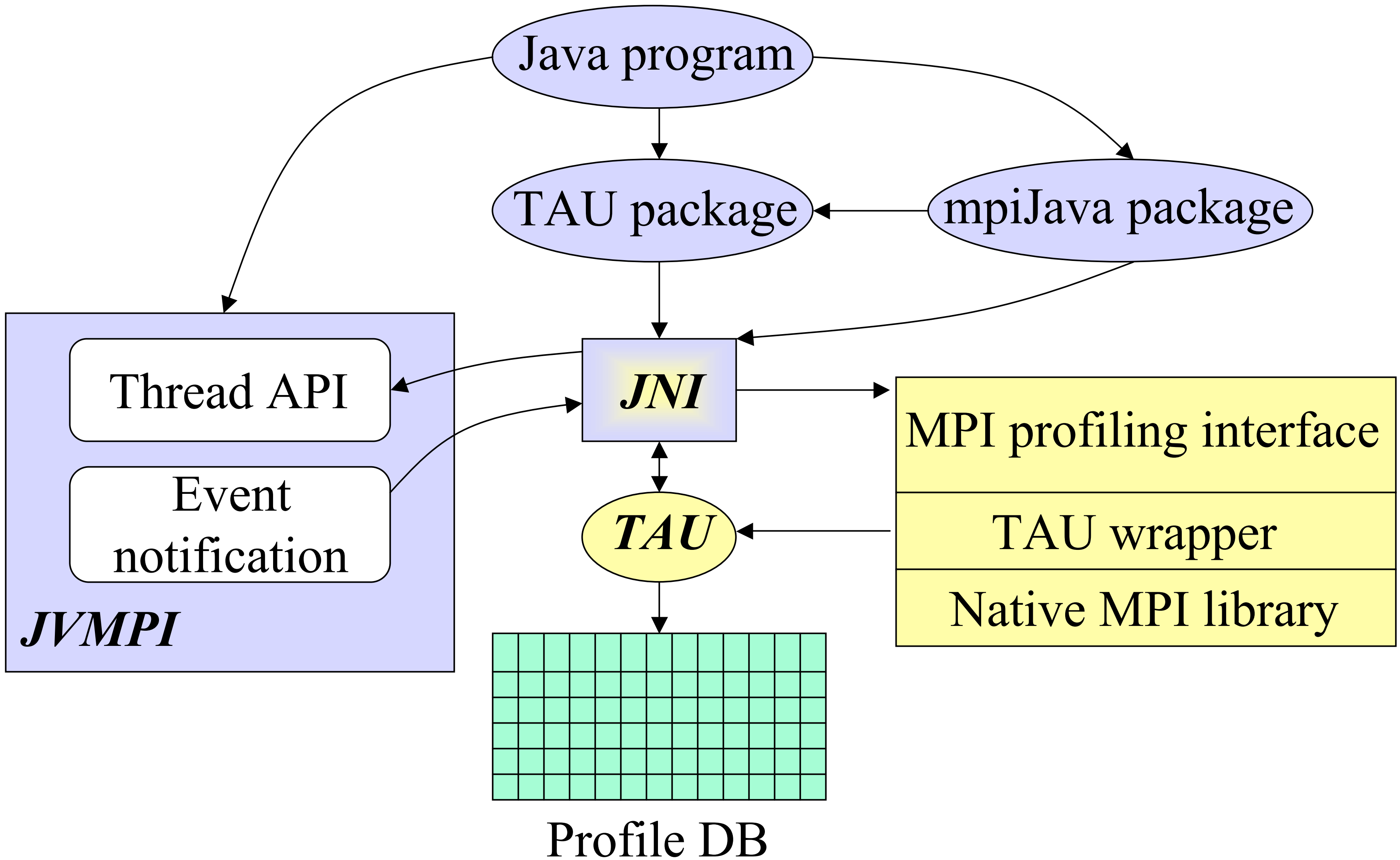
Java Multi-threading and Message Passing

- Java threads and MPI communications
  - Shared-memory multi-threading events
  - Message communications events
- Unified performance measurement and views
  - Integration of performance mechanisms
  - Integrated association of performance events
    - thread event and communication events
    - user-defined (source-level) performance events
    - JVM events
- Requires instrumentation and measurement cooperation

Instrumentation and Measurement Cooperation

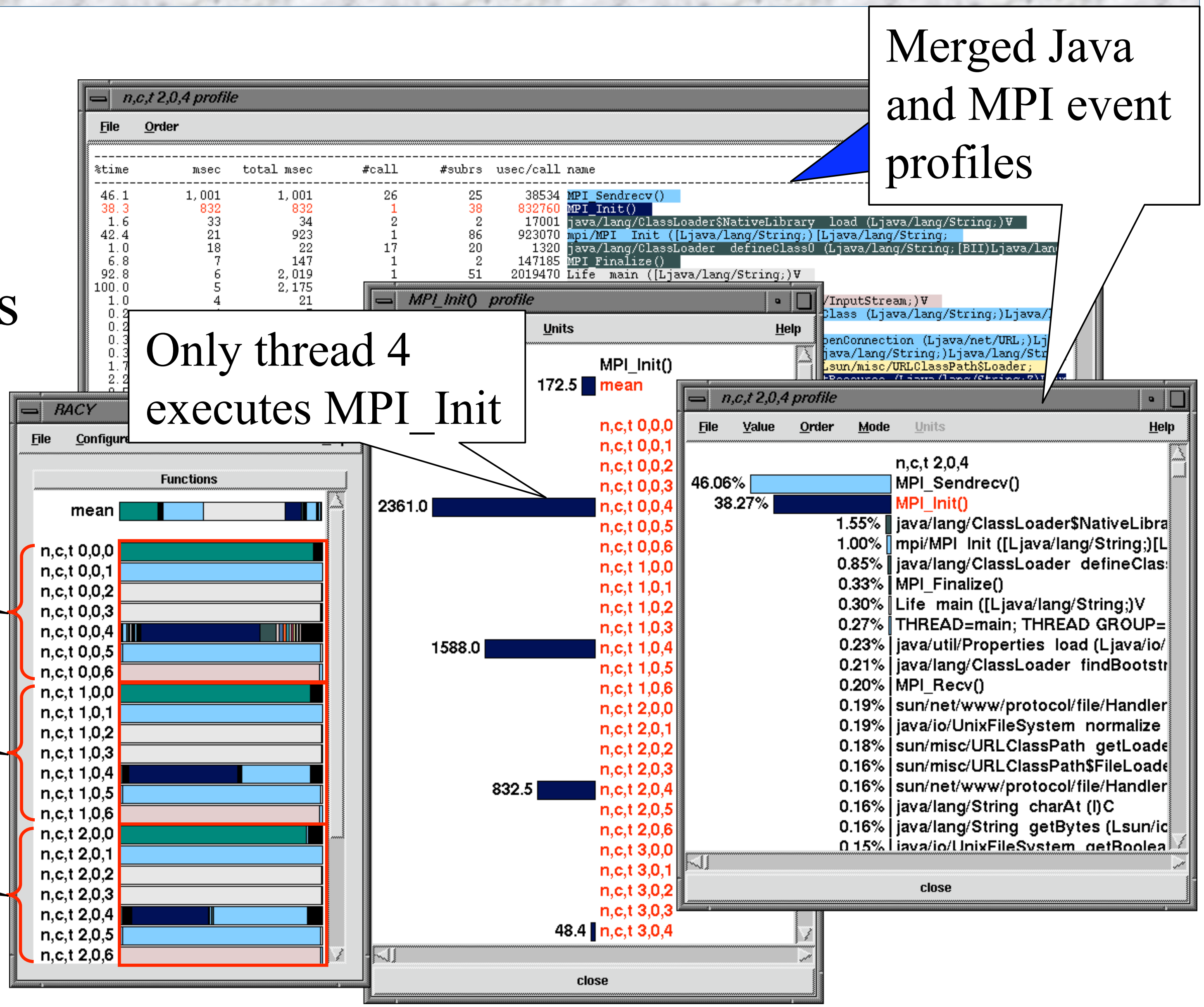
- Problem
  - JVMPI doesn't see MPI events (e.g., rank (node))
  - MPI profiling interfaces doesn't see threads
  - Source instrumentation doesn't see either!
- Need cooperation between interfaces
  - MPI exposes rank and gets thread information
  - JVMPI exposes thread information and gets rank
  - Source instrumentation gets both
  - Post-mortem matching of sends and receives
- Selective instrumentation
  - java -XrunTAU:exclude=java/io,sun

TAU Java Instrumentation Architecture



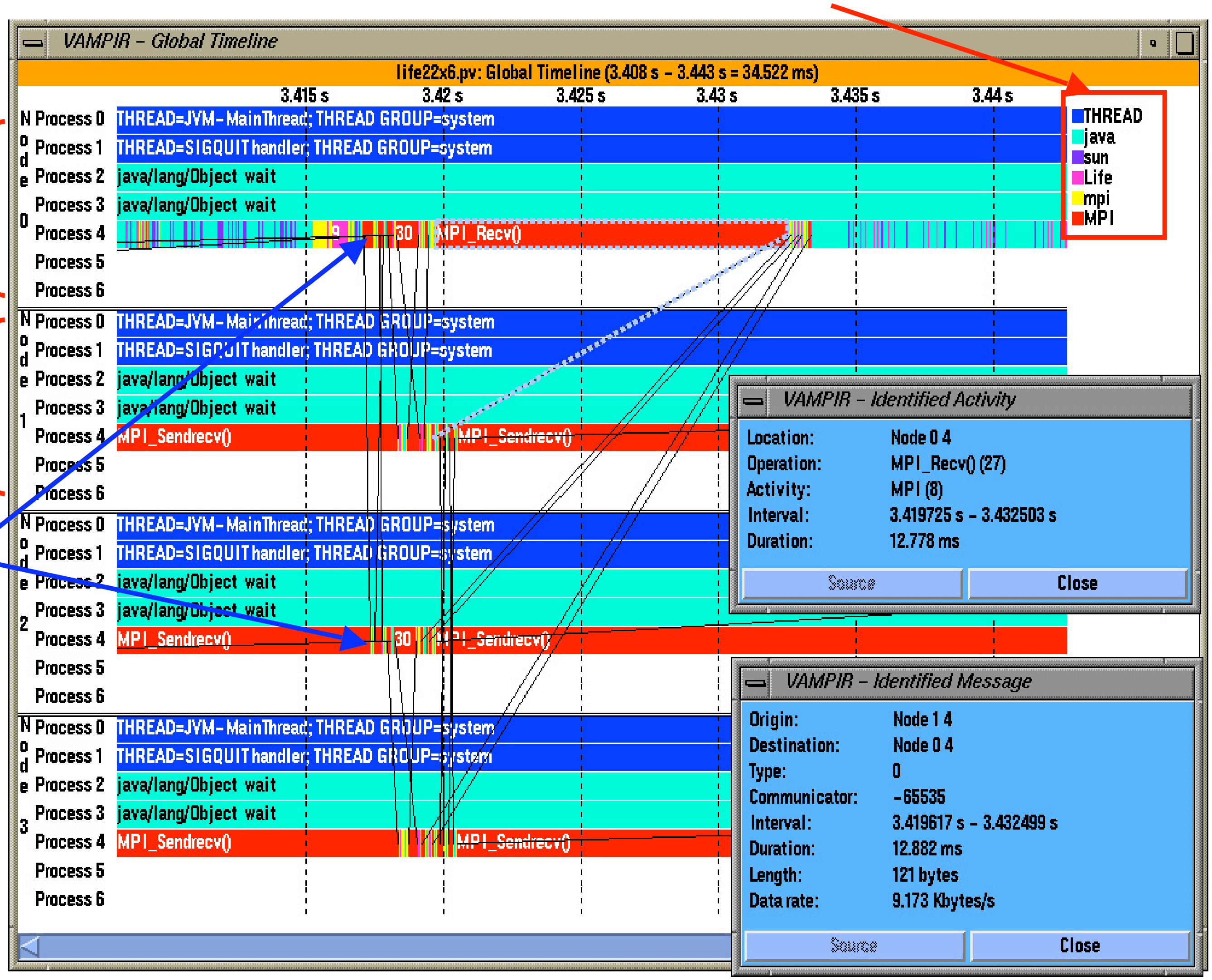
Parallel Java Game of Life (Profile)

- mpiJava testcase
- 4 nodes, 28 threads



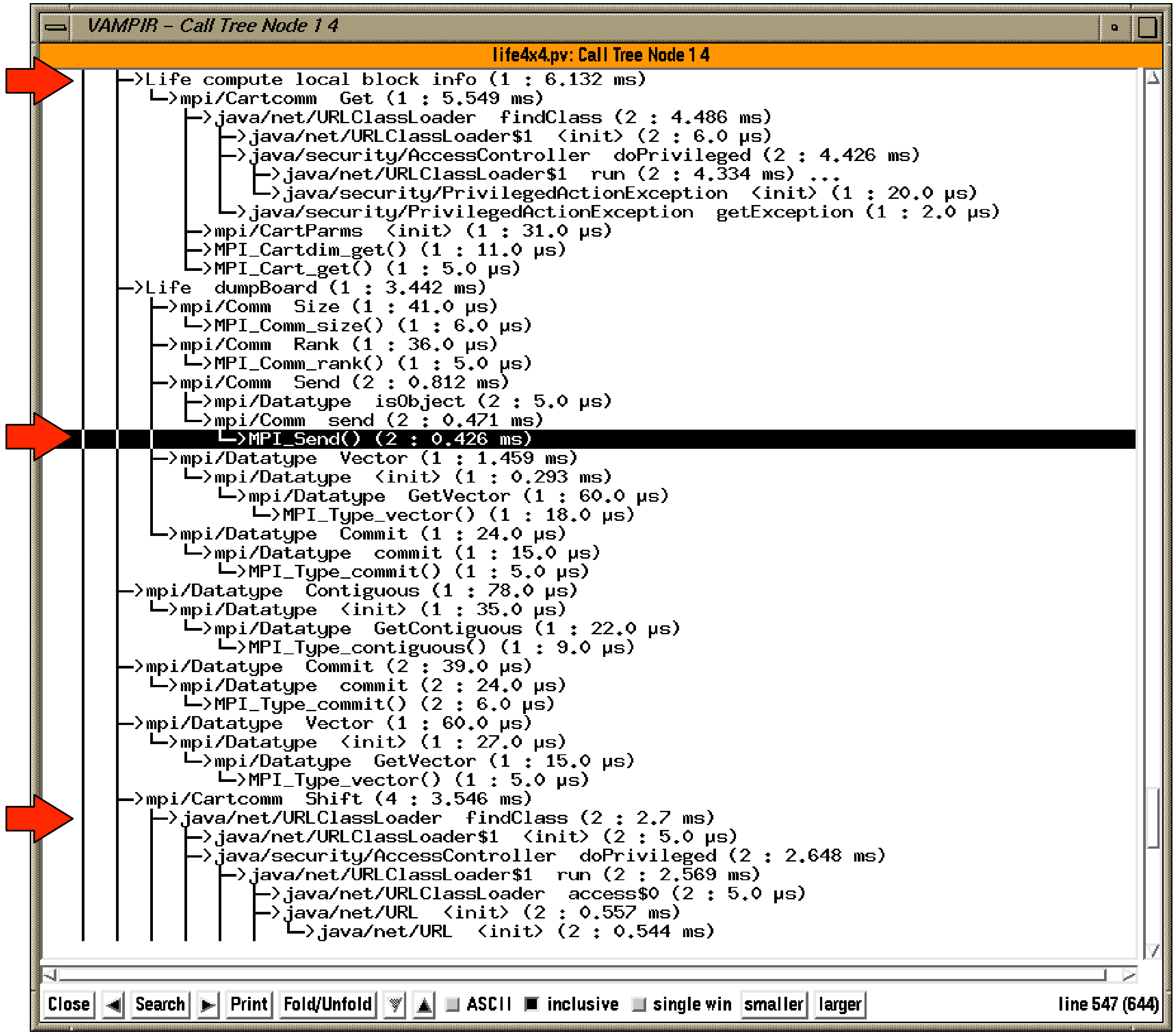
Parallel Java Game of Life (Trace)

- Integrated event tracing
- Multi-level event grouping
- Merged trace viz
- Node process grouping
- Thread message pairing
- Vampir display



Integrated Performance View (Callgraph)

- Source level
- MPI level
- Java packages level

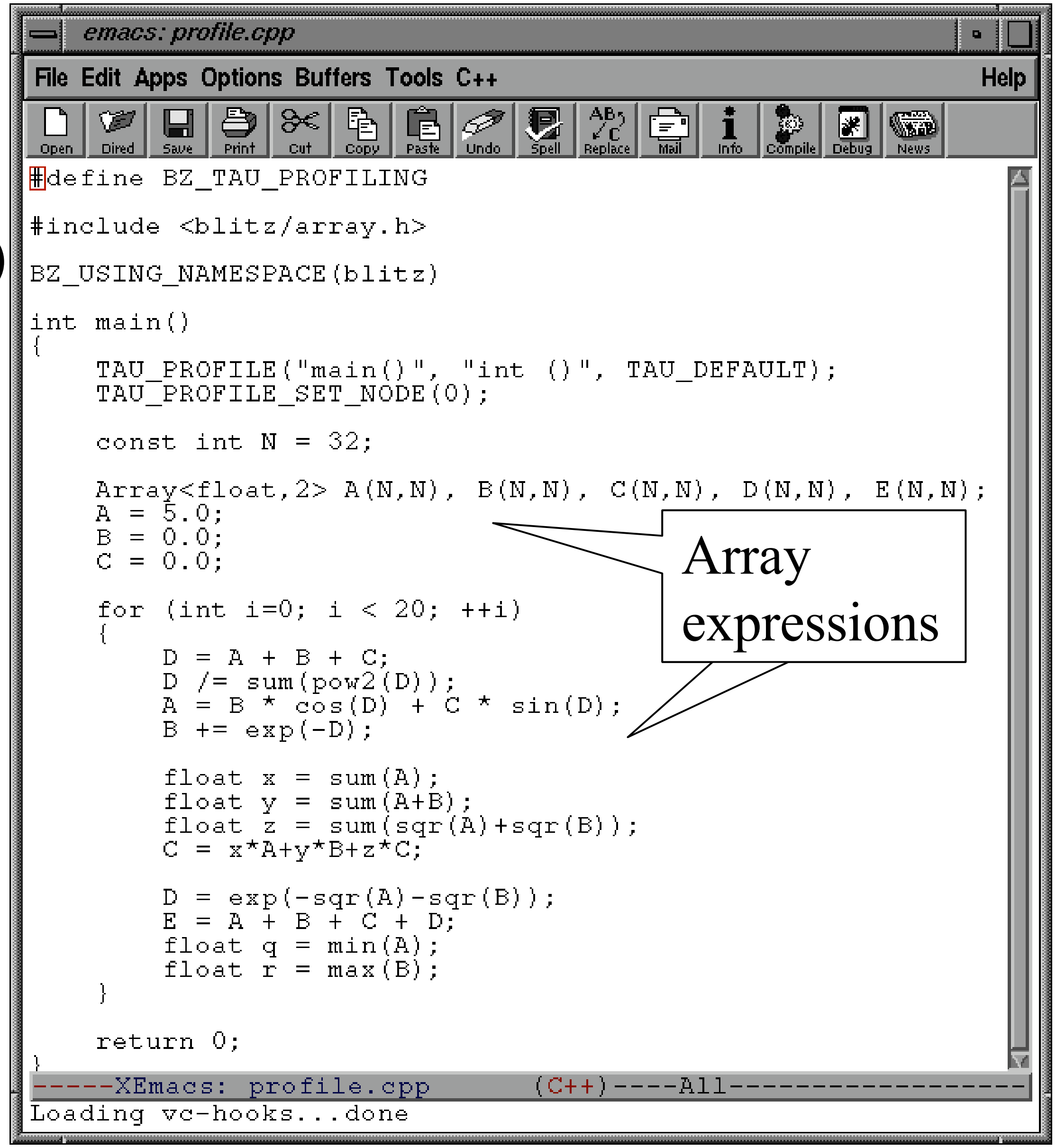


Object-Oriented Programming and C++

- Object-oriented programming is based on concepts of abstract data types, encapsulation, inheritance, …
- Languages (such as C++) provide support implementing domain-specific abstractions in the form of class libraries
- Furthermore, generic programming mechanisms allow for efficient coding abstractions and compile-time transformations
- Creates a semantic gap between the transformed code and what the user expects (and describes in source code)
- Need a mechanism to expose the nature of high-level abstract computation to the performance tools
- Map low-level performance data to high-level semantics

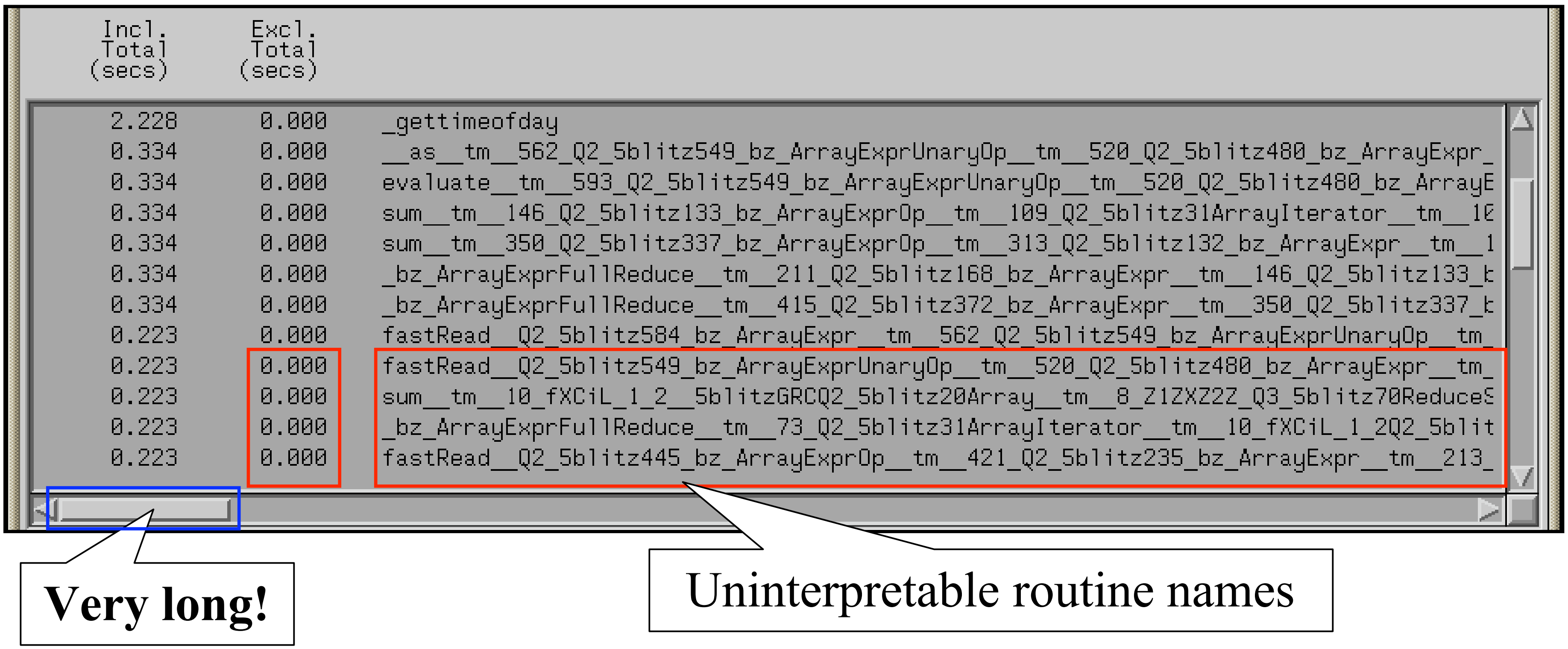
C++ Template Instrumentation (Blitz++, PETE)

- High-level objects
  - Array classes
  - Templates (Blitz++)
- Optimizations
  - Array processing
  - Expressions (PETE)
- Relate performance data to high-level statement
- Complexity of template evaluation



Standard Template Instrumentation Difficulties

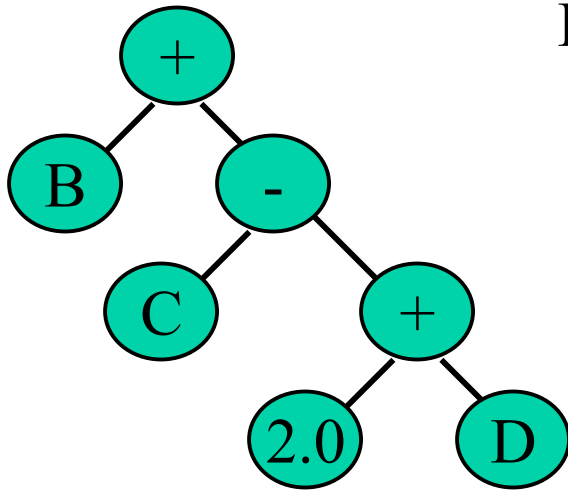
- Instantiated templates result in mangled identifiers
- Standard profiling techniques / tools are deficient
  - Integrated with proprietary compilers
  - Specific systems platforms and programming models



Incl. Total (secs)	Excl. Total (secs)	
2.228	0.000	_gettimeofday
0.334	0.000	__as__tm__562_Q2_5blitz549_bz_ArrayExprUnaryOp__tm__520_Q2_5blitz480_bz_ArrayExpr_
0.334	0.000	evaluate__tm__593_Q2_5blitz549_bz_ArrayExprUnaryOp__tm__520_Q2_5blitz480_bz_ArrayE
0.334	0.000	sum__tm__146_Q2_5blitz133_bz_ArrayExprOp__tm__109_Q2_5blitz31ArrayIterator__tm__1
0.334	0.000	sum__tm__350_Q2_5blitz337_bz_ArrayExprOp__tm__313_Q2_5blitz132_bz_ArrayExpr__tm__1
0.334	0.000	_bz_ArrayExprFullReduce__tm__211_Q2_5blitz168_bz_ArrayExpr__tm__146_Q2_5blitz133_b
0.334	0.000	_bz_ArrayExprFullReduce__tm__415_Q2_5blitz372_bz_ArrayExpr__tm__350_Q2_5blitz337_b
0.223	0.000	fastRead__Q2_5blitz584_bz_ArrayExpr__tm__562_Q2_5blitz549_bz_ArrayExprUnaryOp__tm_
0.223	0.000	fastRead__Q2_5blitz549_bz_ArrayExprUnaryOp__tm__520_Q2_5blitz480_bz_ArrayExpr__tm_
0.223	0.000	sum__tm__10_fXCiL_1_2__5blitzGRCQ2_5blitz20Array__tm__8_Z1ZXZ2Z_Q3_5blitz70ReduceS
0.223	0.000	_bz_ArrayExprFullReduce__tm__73_Q2_5blitz31ArrayIterator__tm__10_fXCiL_1_2Q2_5blit
0.223	0.000	fastRead__Q2_5blitz445_bz_ArrayExprOp__tm__421_Q2_5blitz235_bz_ArrayExpr__tm__213_

Blitz++ Library Instrumentation

- Expression templates embed the form of the expression in a template name



BinOp<Add,
B, <BinOp<Subtract,
C, <BinOp<Multiply,
Scalar<2.0>, D>>>

- In Blitz++, the library describes the structure of the expression template to the profiling toolkit
- Allows for pretty printing the expression templates

Expression: B + C - 2.0 * D

Blitz++ Library Instrumentation (example)

```
#ifdef BZ_TAU_PROFILING
static string exprDescription;
if (!exprDescription.length()) {
  exprDescription = "A";
  prettyPrintFormat format(_bz_true); // Terse mode on
  format.nextArrayOperandSymbol();
  T_update::prettyPrint(exprDescription);
  expr.prettyPrint(exprDescription, format);
}
TAU_PROFILE(" ", exprDescription, TAU_BLITZ);
#endif
```

TAU Instrumentation and Profiling for C++

TAU and SMARTS: Asynchronous Performance

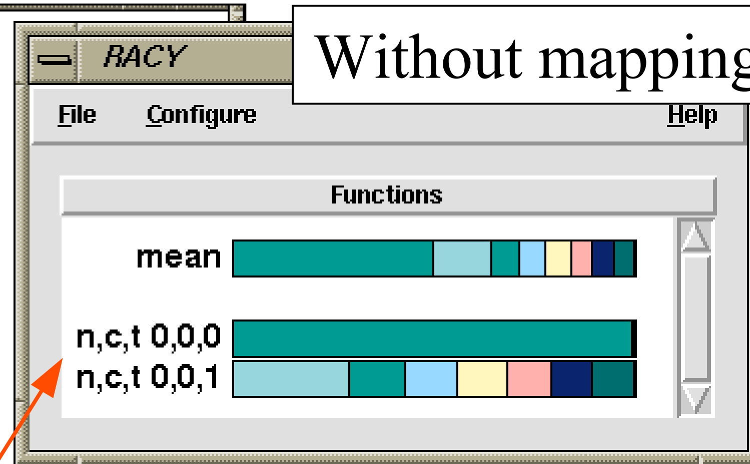
- Scalable Multithreaded Asynchronous RTS
  - User-level threads, light-weight virtual processors
  - Macro-dataflow, asynchronous execution interleaving iterates from data-parallel statements
  - Integrated with POOMA II (parallel dense array library)
- Measurement of asynchronous parallel execution
  - Utilized the TAU mapping API
  - Associate iterate performance with data parallel statement
  - Evaluate different scheduling policies
- *"SMARTS: Exploting Temporal Locality and Parallelism through Vertical Execution" (ICS '99)*

TAU Mapping of Asynchronous Execution

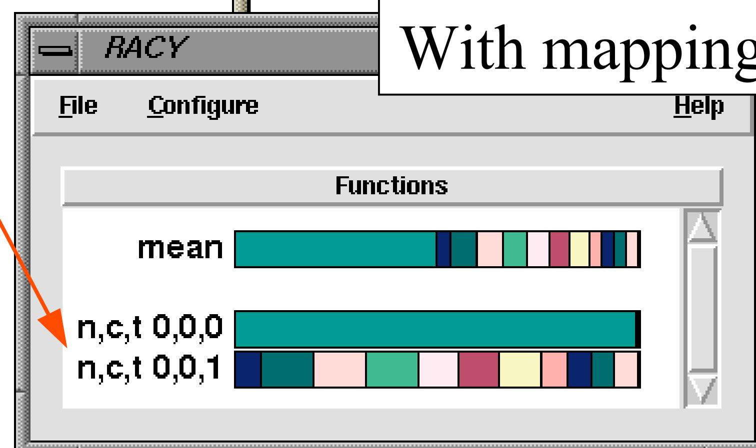
```
#include "Pooma/Arrays.h"

#include <iostream.h>

// The size of each side of the domain.
const int N = 3*1024;

int
main(
    int                    argc,          // argument count
    char *                 argv[]         // argument list
){
    // Initialize Pooma.
    Pooma::initialize(argc, argv);

    // The array we'll be solving for
    Array<2> A(N, N), B(N,N), C(N,N), D(N,N), E(N,N);

    // Must block since we're doing some scalar code (see Tutorial 4).
    Pooma::blockAndEvaluate();

    A = 1.0;
    B = 2.0;
    C = 3.0;
    D = 4.0;
    E = 5.0;

    A = B + C + D;
    C = E - A + 2.0 * B;
    D = A + C;
    C = D + A - B;
    A = 2.0 * D + E ;
    E = 1.5 * B - A ;

    Pooma::blockAndEvaluate();

    cout << "D(1,1) = " << D(1,1) << endl;
    cout << "D(9,9) = " << D(9,9) << endl;

    // Clean up Pooma and report success.
    Pooma::finalize();
    return 0;
}
```

Two threads executing

Without mapping

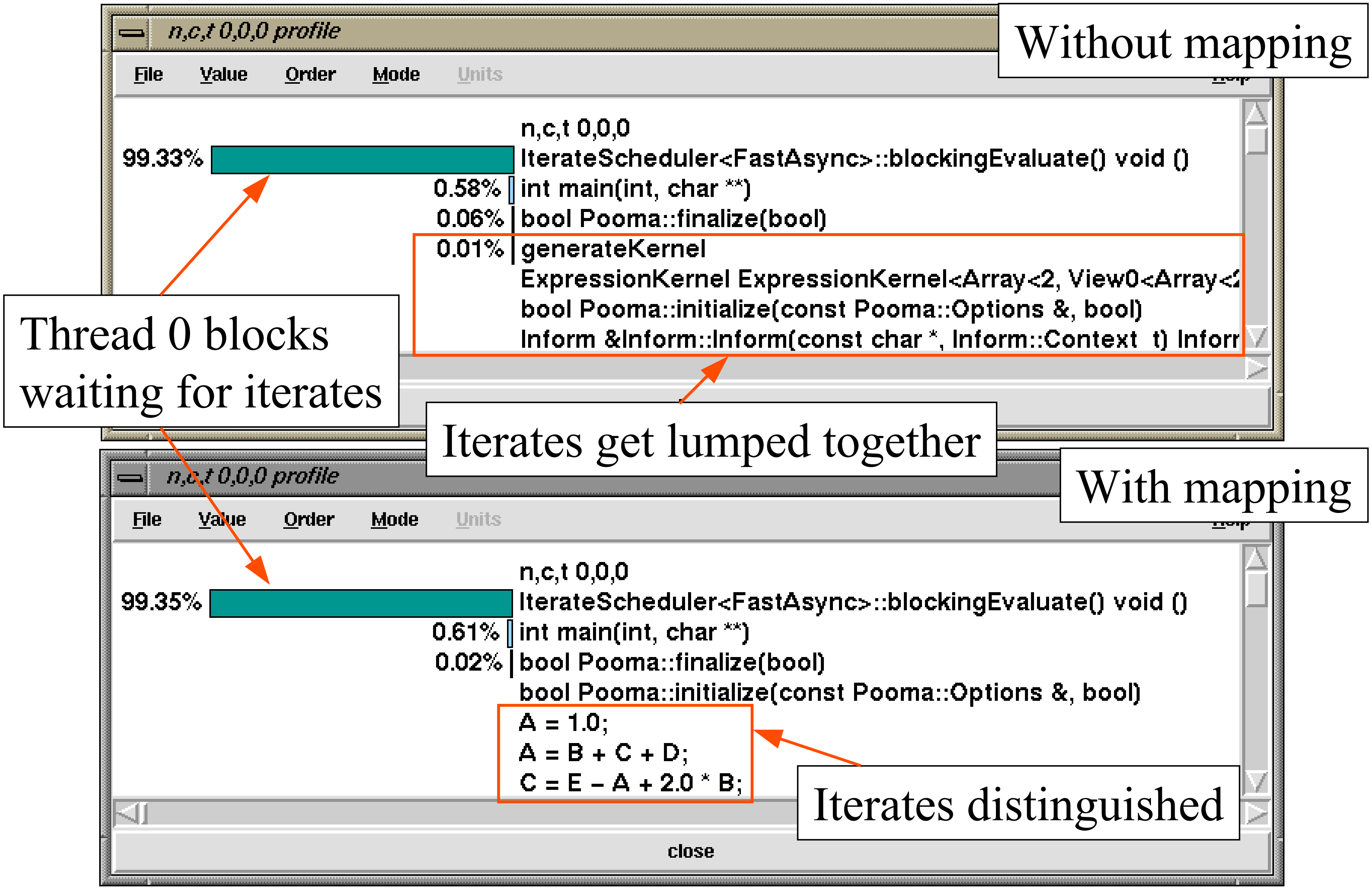


With mapping



POOMA / SMARTS

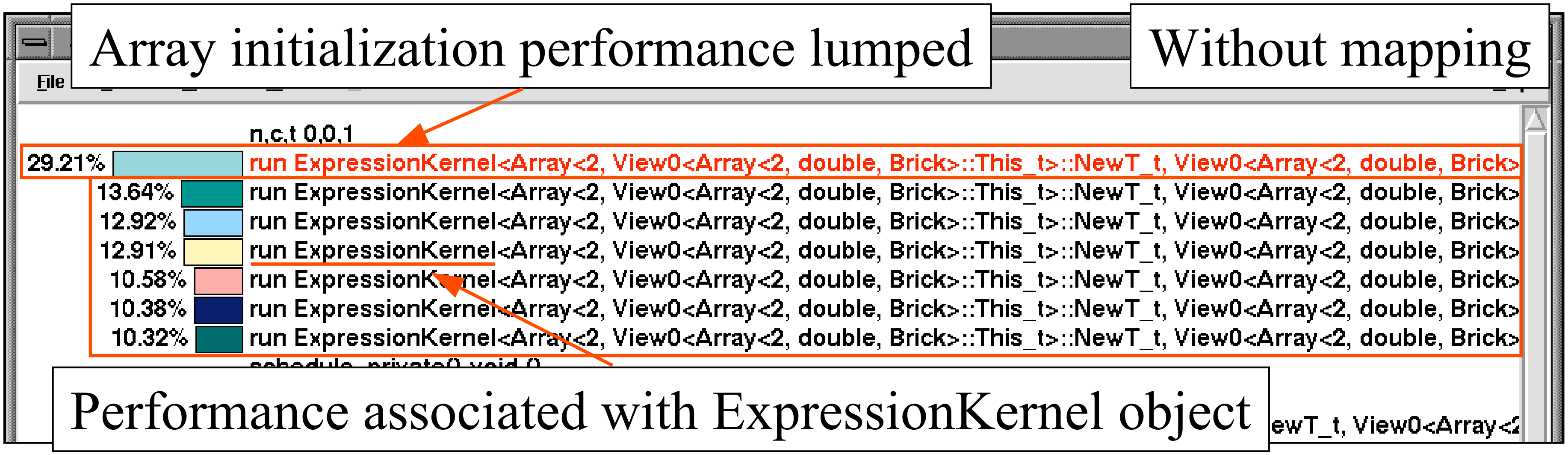
With and Without Mapping (Thread 0)



With and Without Mapping (Thread 1)

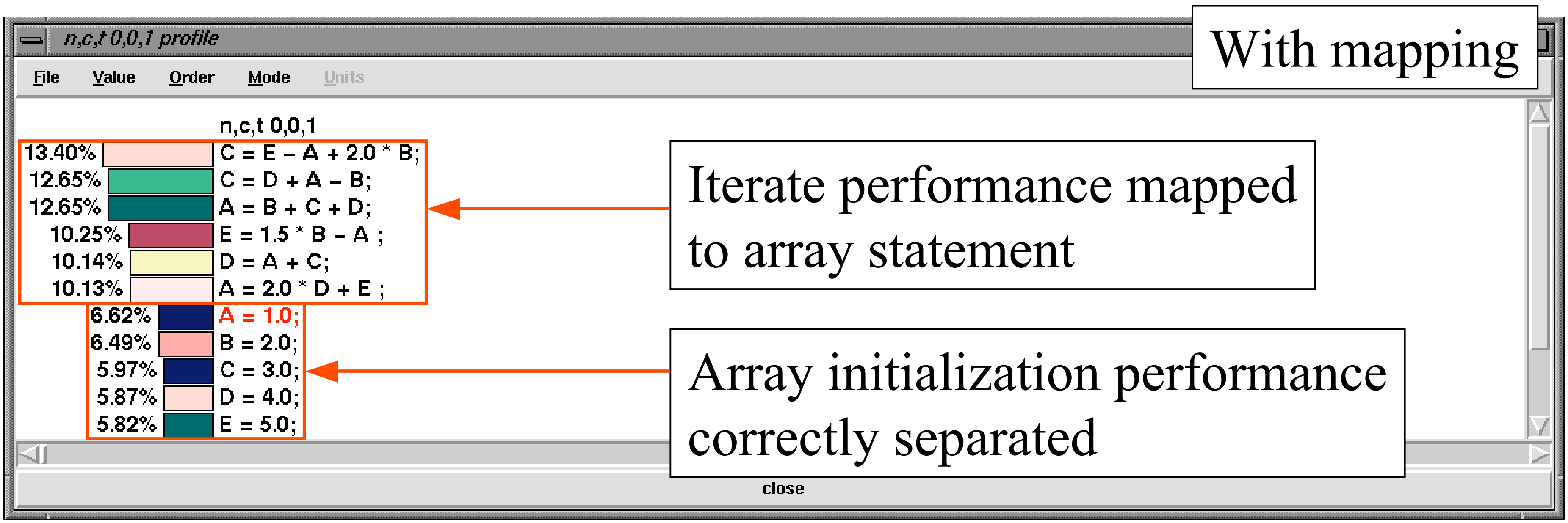
Array initialization performance lumped

Without mapping



Performance associated with ExpressionKernel object

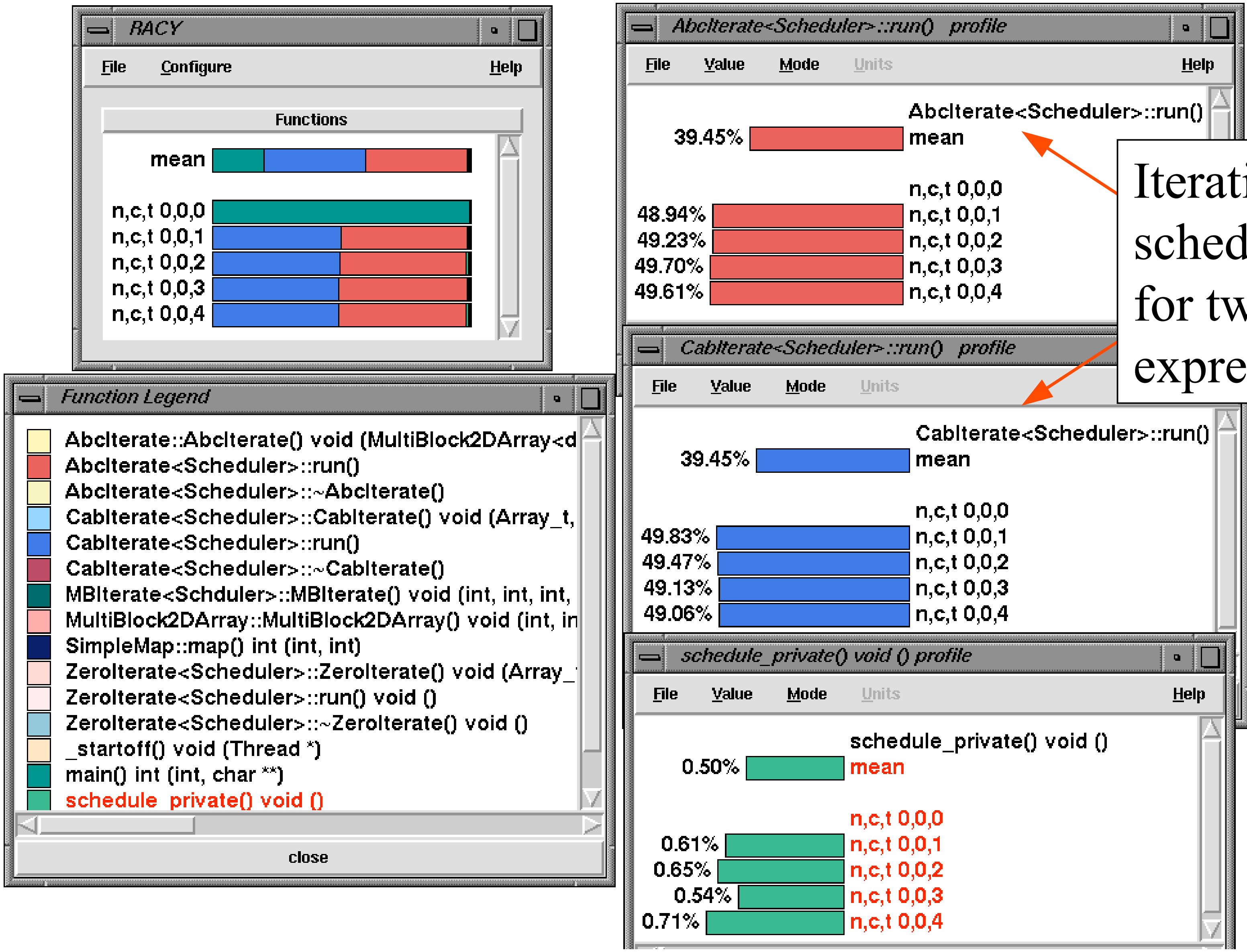
With mapping



Iterate performance mapped to array statement

Array initialization performance correctly separated

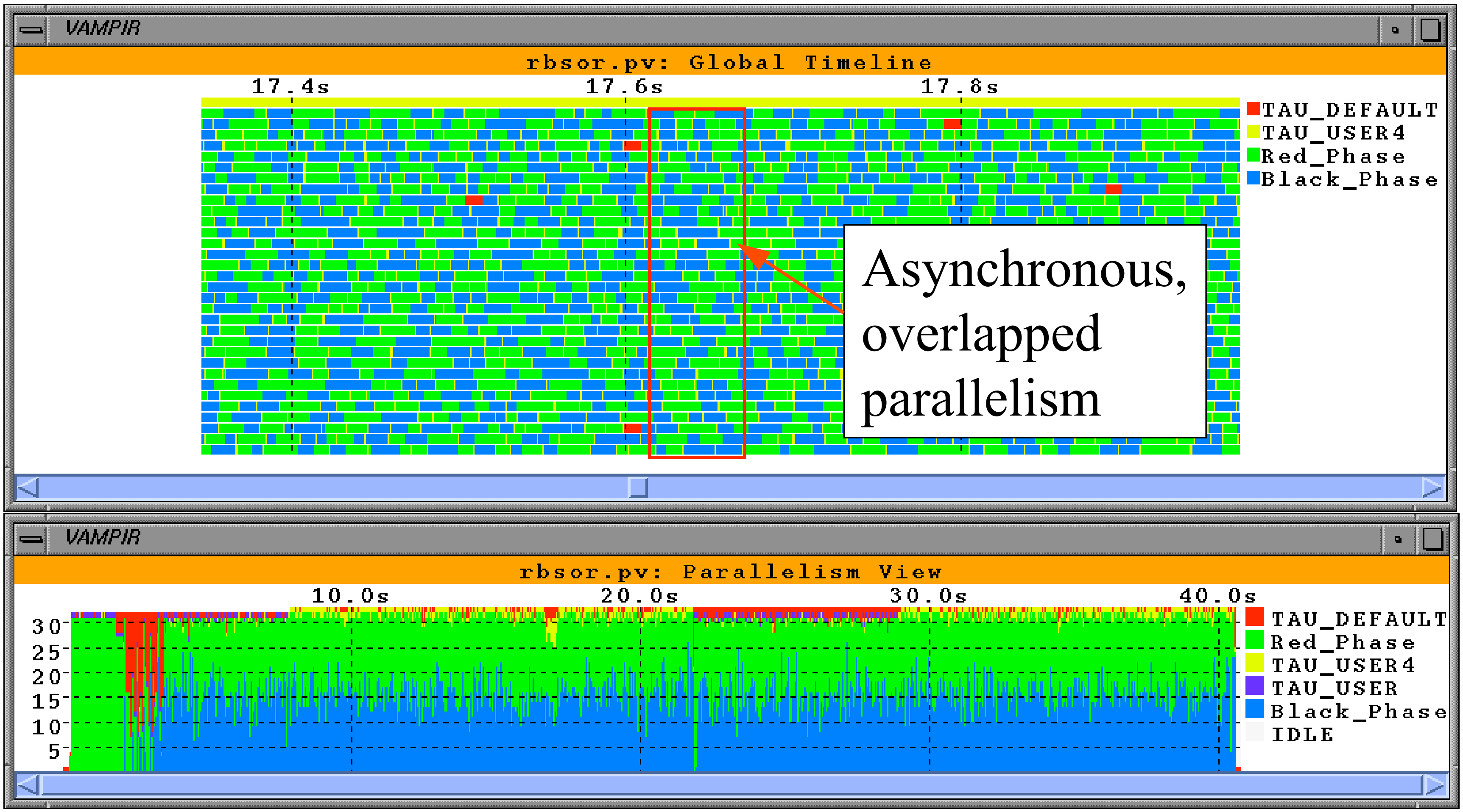
TAU Profiling of SMARTS Scheduling



Iteration scheduling for two array expressions

SMARTS Tracing (SOR) – Vampir Visualization

- SCVE scheduler used in Red/Black SOR running on 32 processors of SGI Origin 2000



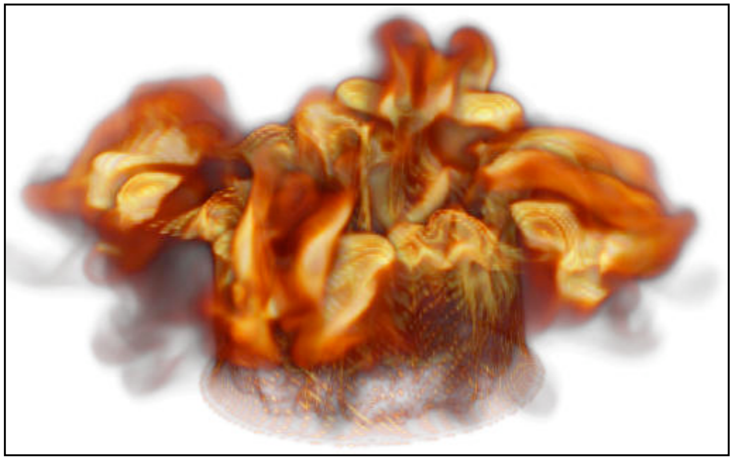
Scientific Software (Performance) Engineering

- Modern scientific simulation software is complex (⇑)
  - Large development teams of diverse expertise
  - Simultaneous development on different system parts
  - Iterative, multi-stage, long-term software development
- Need support for managing complex software process
  - Software engineering tools for revision control, automated testing, and bug tracking are commonplace
  - In contrast, tools for performance engineering are not
    - evaluation (measurement, analysis, benchmarking)
    - optimization (diagnosis, tracking, prediction, tuning)
- Incorporate performance engineering methodology and support by flexible and robust performance tools

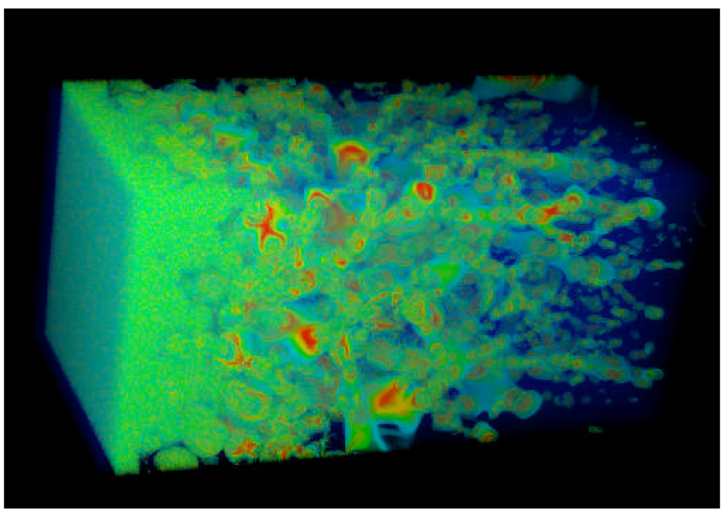
Hierarchical Parallel Software (C-SAFE/Uintah)

- Center for Simulation of Accidental Fires & Explosions
  - ASCI Level 1 center, University of Utah
  - PSE for multi-model simulation high-energy explosion
  - Combine fundamental chemistry and engineering physics
  - Integrate non-linear solvers, optimization, computational steering, visualization, and experimental data verification
  - Support very large-scale coupled simulations
- Computer science problems:
  - Coupling multiple scientific simulation codes with different numerical and software properties
  - Software engineering across diverse expert teams
  - Achieving high performance on large-scale systems

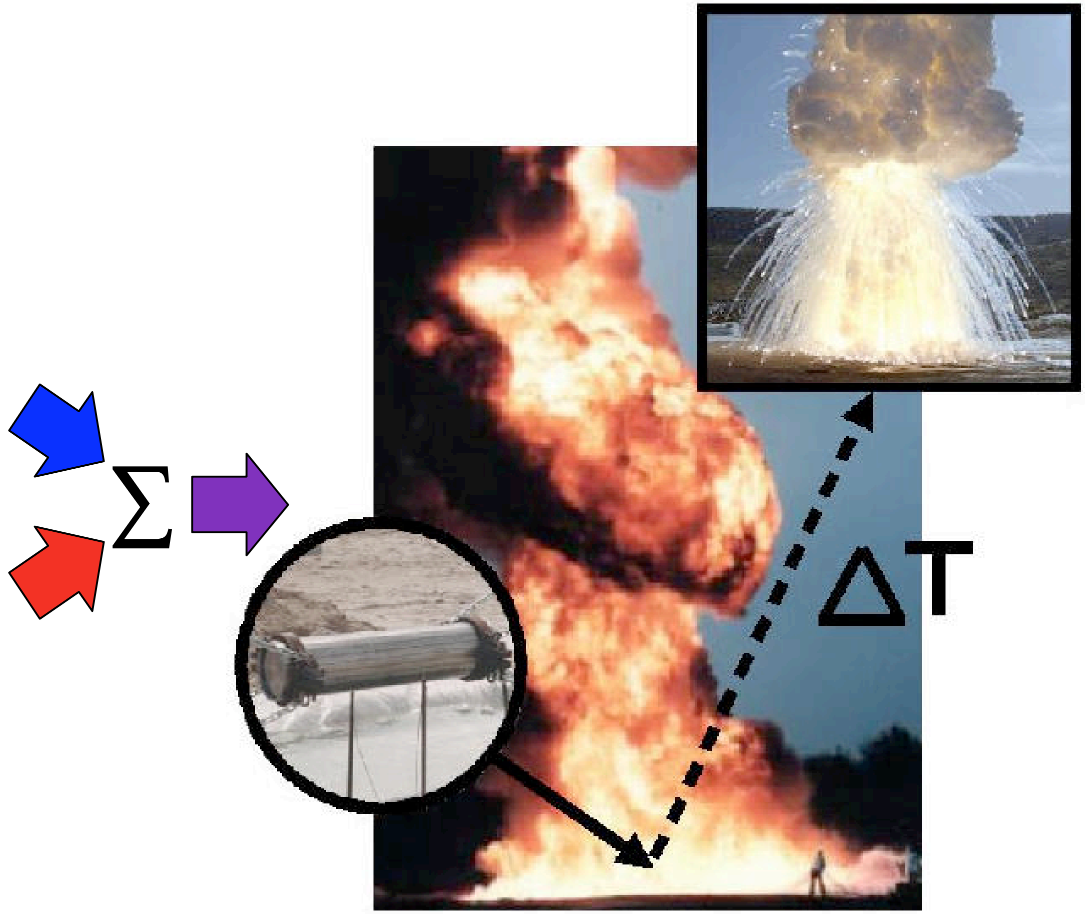
Example C-SAFE Simulation Problems



Heptane fire simulation



Material stress simulation

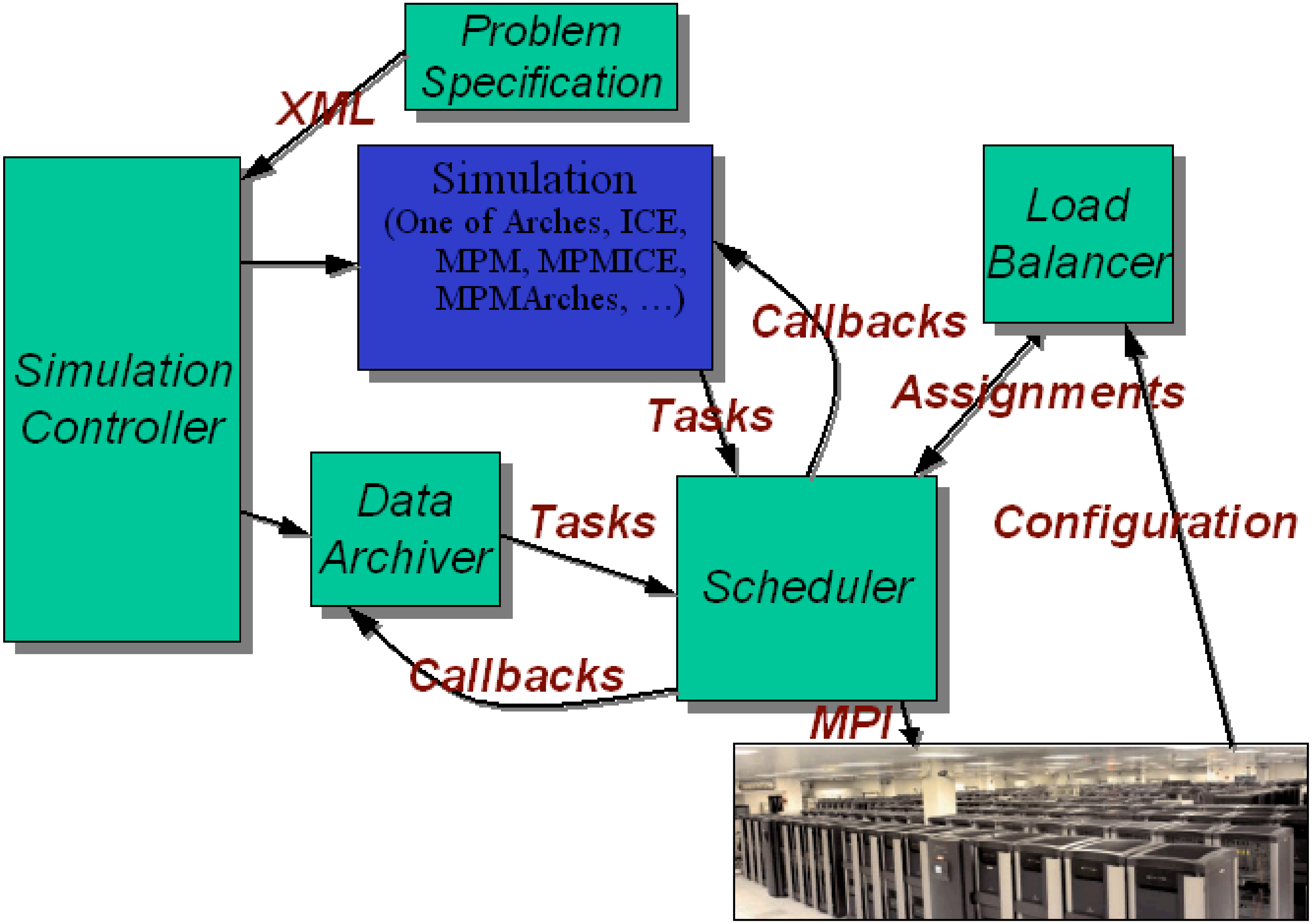


Typical C-SAFE simulation with a billion degrees of freedom and non-linear time dynamics

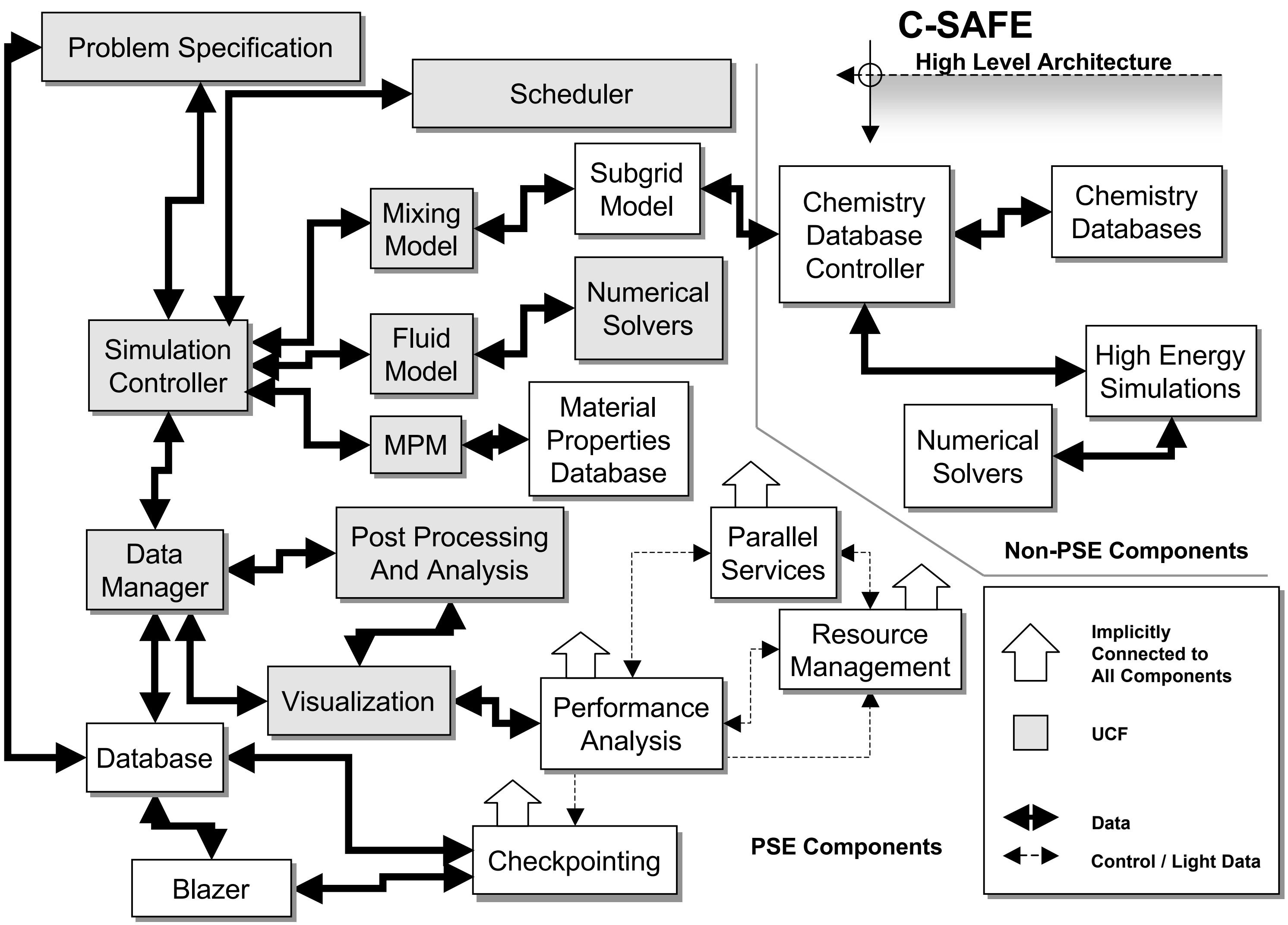
Uintah Problem Solving Environment

- Uintah parallel programming framework
  - Component-based and object-parallel
  - Multi-model task-graph scheduling and execution
  - Shared-memory (thread), distributed-memory (MPI), and mixed-model parallelization
- Design and implement Uintah component architecture
  - Application programmers provide
    - description of computation (tasks and variables)
    - code to perform task on sub-region of space (patch)
  - Components for scheduling, partitioning, load balance, …
  - Follow Common Component Architecture (CCA) model
- Design and implement Uintah Computational Framework (UCF) on top of the component architecture

Uintah High-Level Component View



Uintah Parallel Component Architecture

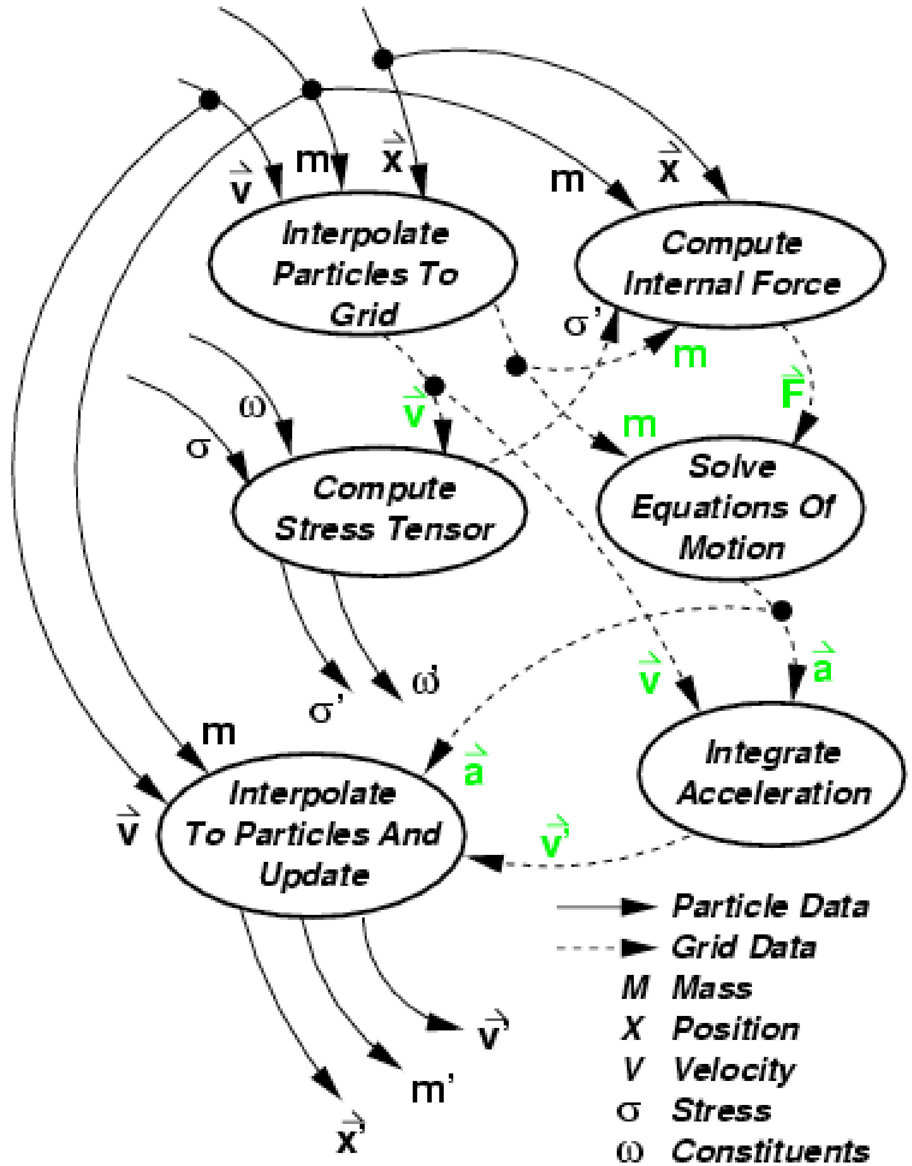


Uintah Computational Framework

- Execution model based on software (macro) dataflow
  - Exposes parallelism and hides data transport latency
  - Computations expressed a directed acyclic graphs of tasks
    - consumes input and produces output (input to future task)
    - input/outputs specified for each patch in a structured grid
- Abstraction of global single-assignment memory
  - *DataWarehouse*
  - Directory mapping names to values (array structured)
  - Write value once then communicate to awaiting tasks
- Task graph gets mapped to processing resources
- Communications schedule approximates global optimal

Uintah Task Graph (Material Point Method)

- Diagram of named tasks (ovals) and data (edges)
- Imminent computation
  - Dataflow-constrained
- MPM
  - Newtonian material point motion time step
  - Solid: values defined at material point (particle)
  - Dashed: values defined at vertex (grid)
  - Prime (‘): values updated during time step



Uintah PSE

- UCF automatically sets up:
  - Domain decomposition
  - Inter-processor communication with aggregation/reduction
  - Parallel I/O
  - Checkpoint and restart
  - Performance measurement and analysis (stay tuned)
- Software engineering
  - Coding standards
  - CVS (Commits: Y3 - 26.6 files/day, Y4 - 29.9 files/day)
  - Correctness regression testing with bugzilla bug tracking
  - Nightly build (parallel compiles)
  - 170,000 lines of code (Fortran and C++ tasks supported)

Performance Technology Integration

- Uintah presents challenges to performance integration
  - Software diversity and structure
    - UCF middleware, simulation code modules
    - component-based hierarchy
  - Portability objectives
    - cross-language (C, C++, F90) and cross-platform
    - multi-parallelism: thread, message passing, mixed
  - Scalability objectives
  - High-level programming and execution abstractions
- Requires flexible and robust performance technology
- Requires support for performance mapping

Performance Analysis Objectives for Uintah

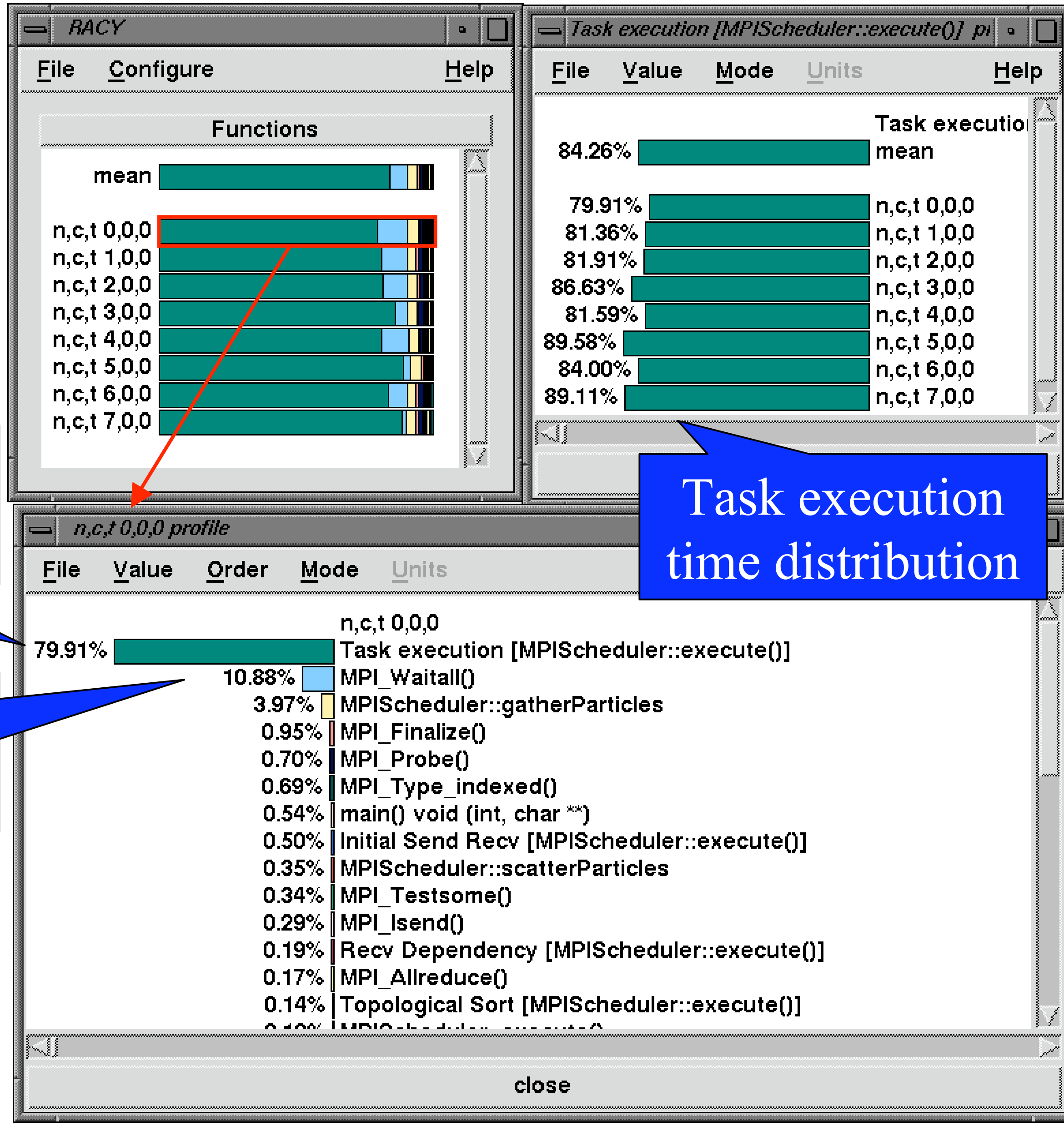
- Micro tuning
  - Optimization of simulation code (task) kernels for maximum serial performance
- Scalability tuning
  - Identification of parallel execution bottlenecks
    - overheads: scheduler, data warehouse, communication
    - load imbalance
  - Adjustment of task graph decomposition and scheduling
- Performance tracking
  - Understand performance impacts of code modifications
  - Throughout course of software development
    - C-SAFE application and UCF software

Uintah Performance Engineering Approach

- Contemporary performance methodology focuses on control flow (function) level measurement and analysis
- C-SAFE application involves coupled-models with task-based parallelism and dataflow control constraints
- Performance engineering on *algorithmic* (task) basis
  - Observe performance based on algorithm (task) semantics
  - Analyze task performance characteristics in relation to other simulation tasks and UCF components
    - scientific component developers can concentrate on performance improvement at algorithmic level
    - UCF developers can concentrate on bottlenecks not directly associated with simulation module code

Task Execution in Uintah Parallel Scheduler

- Profile methods and functions in scheduler and in MPI library



Task execution time dominates (what task?)

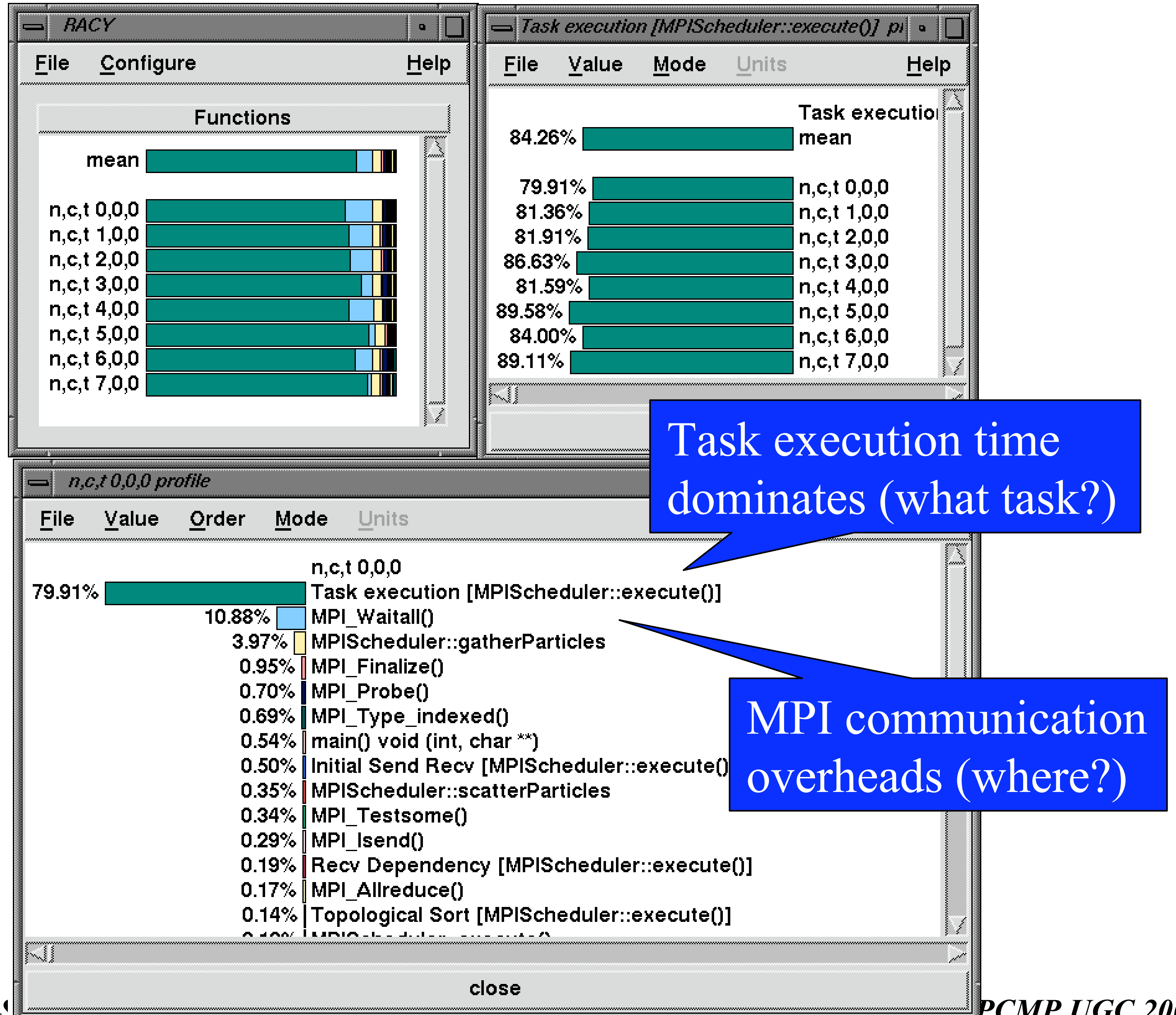
MPI communication overheads (where?)

Task execution time distribution

- Need to *map* performance data!

Task Execution in Uintah Parallel Scheduler

- Profiling methods and functions
- Need to map the performance data



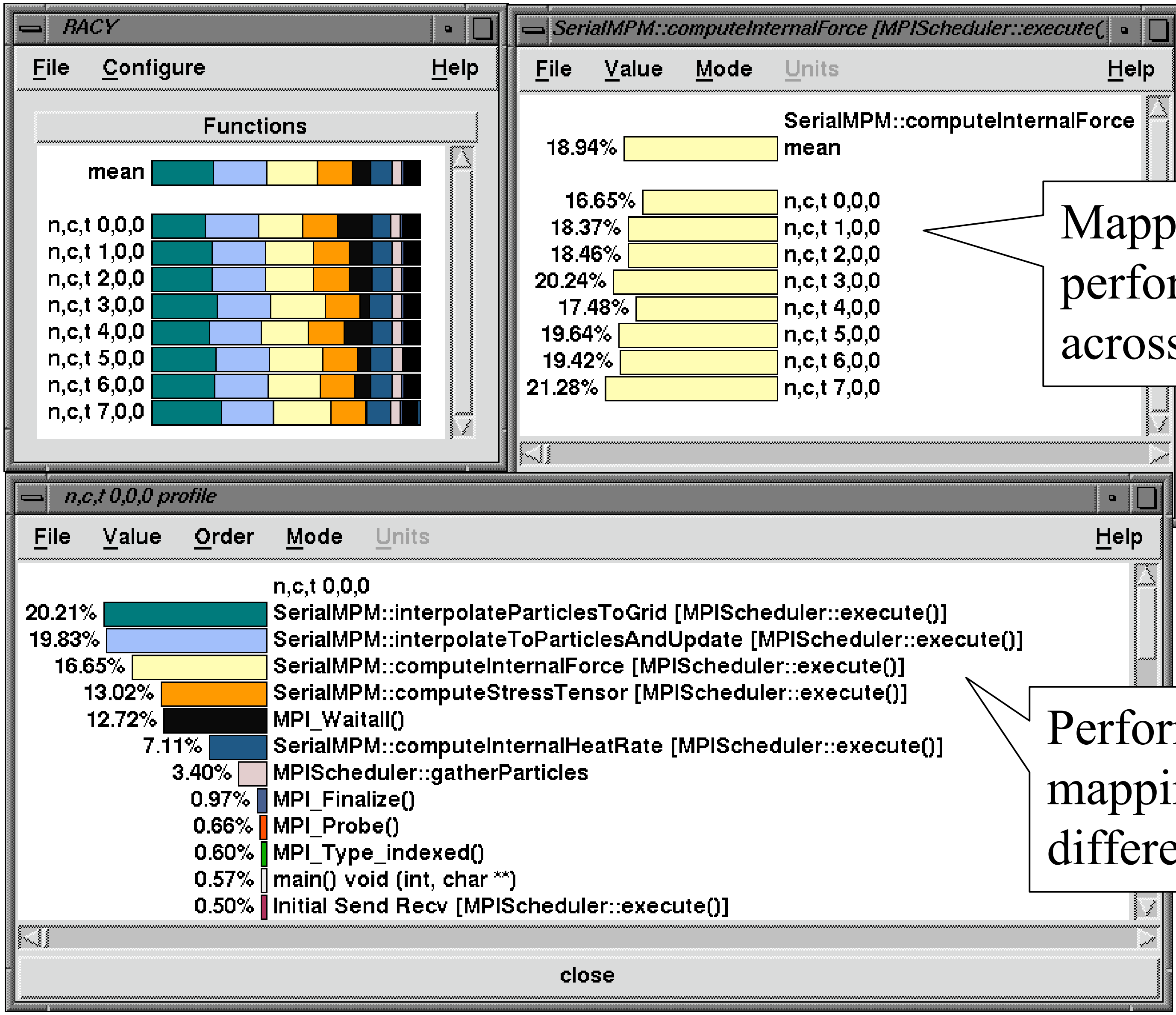
Task Computation and Mapping

- Task computations on individual particles generate work packets that are scheduled and executed
  - Work packets that "interpolate particles to grid"
- Assign semantic name to a task abstraction
  - SerialMPM::interpolateParticleToGrid
- Partition execution time among different tasks
  - Need to relate the performance of each particle computation (work packet) to the associated task
  - Map TAU timer object to task (abstract) computation
  - Mapping: task object ⇔ grid object ⇔ patch objects
- Partition performance data on domain-specific axes
- Helps bridge the semantic-gap!

Mapping Instrumentation (example)

```
void MPIScheduler::execute(const ProcessorGroup * pc,
                           DataWarehouseP   & old_dw,
                           DataWarehouseP   & dw ) {
  ...
  TAU_MAPPING_CREATE(
    task->getName(), "[MPIScheduler::execute()]",
    (TauGroup_t)(void*)task->getName(), task->getName(), 0);
  ...
  TAU_MAPPING_OBJECT(tautimer)
  TAU_MAPPING_LINK(tautimer,(TauGroup_t)(void*)task->getName());
  // EXTERNAL ASSOCIATION
  ...
  TAU_MAPPING_PROFILE_TIMER(doitprofiler, tautimer, 0)
  TAU_MAPPING_PROFILE_START(doitprofiler,0);
  task->doit(pc);
  TAU_MAPPING_PROFILE_STOP(0);
  ...
}
```

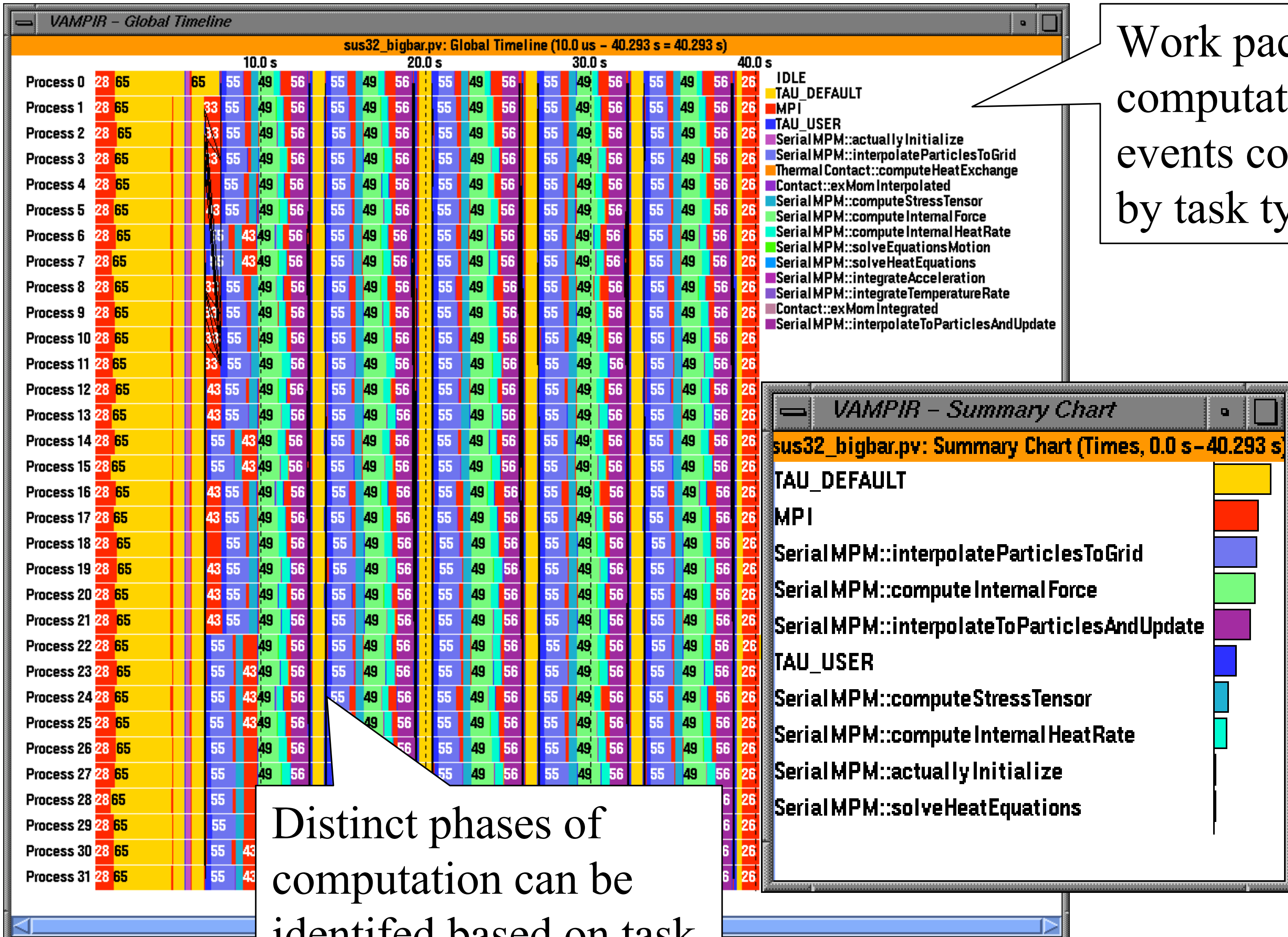
Work Packet – to – Task Mapping (Profile)



Mapped task performance across processes

Performance mapping for different tasks

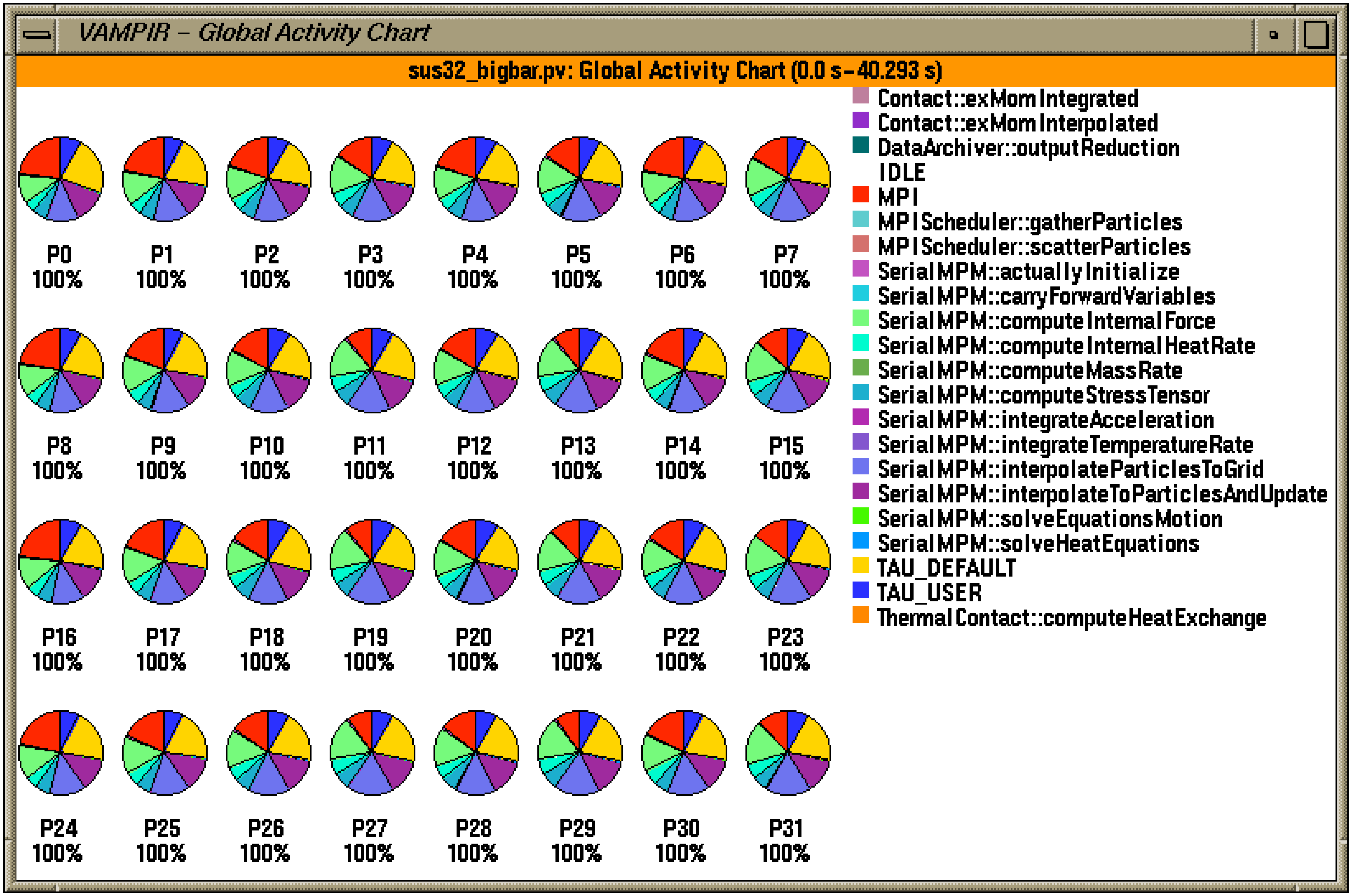
Work Packet – to – Task Mapping (Trace)



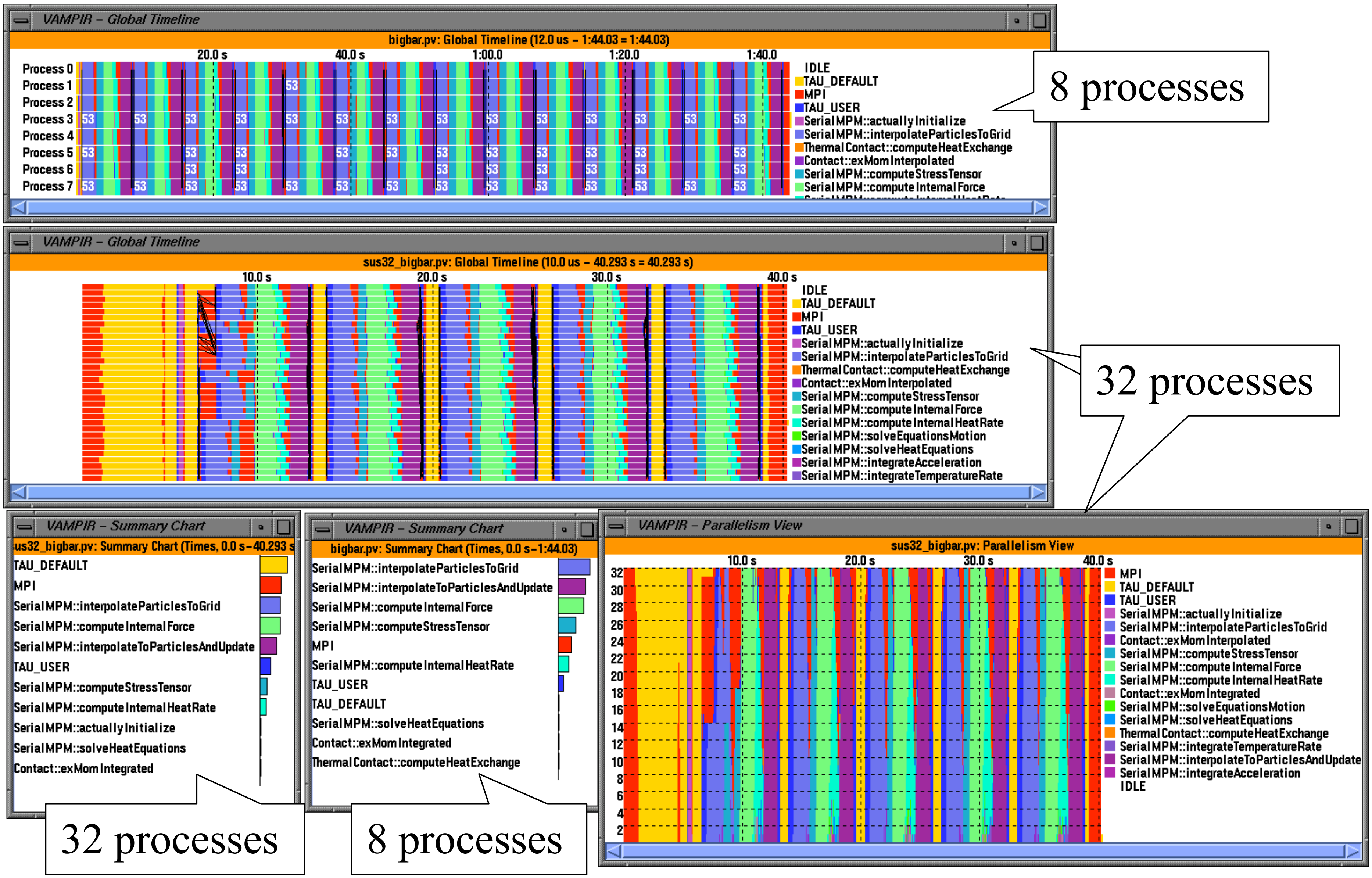
Work packet computation events colored by task type

Distinct phases of computation can be identifed based on task

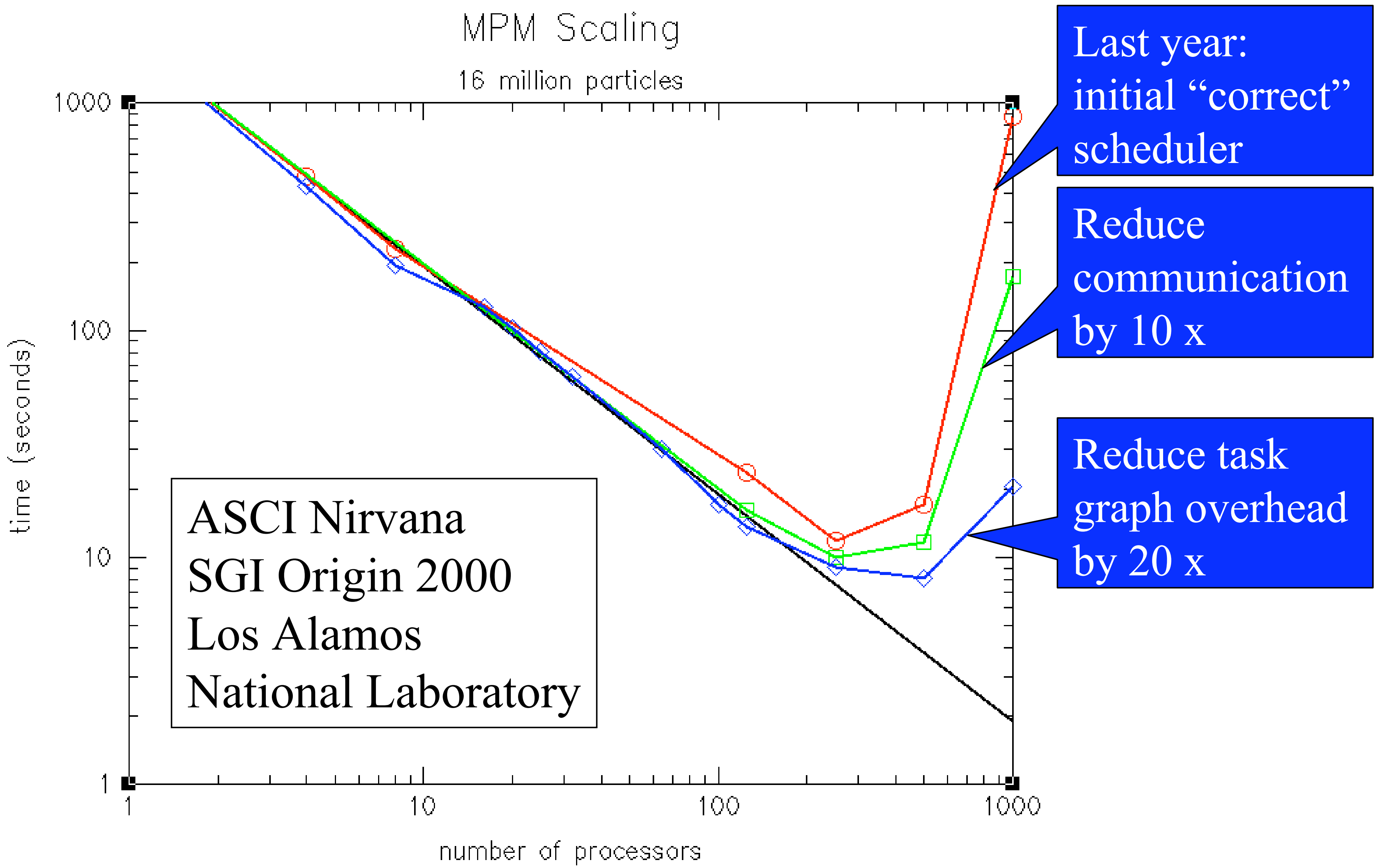
Statistics for Relative Task Contributions



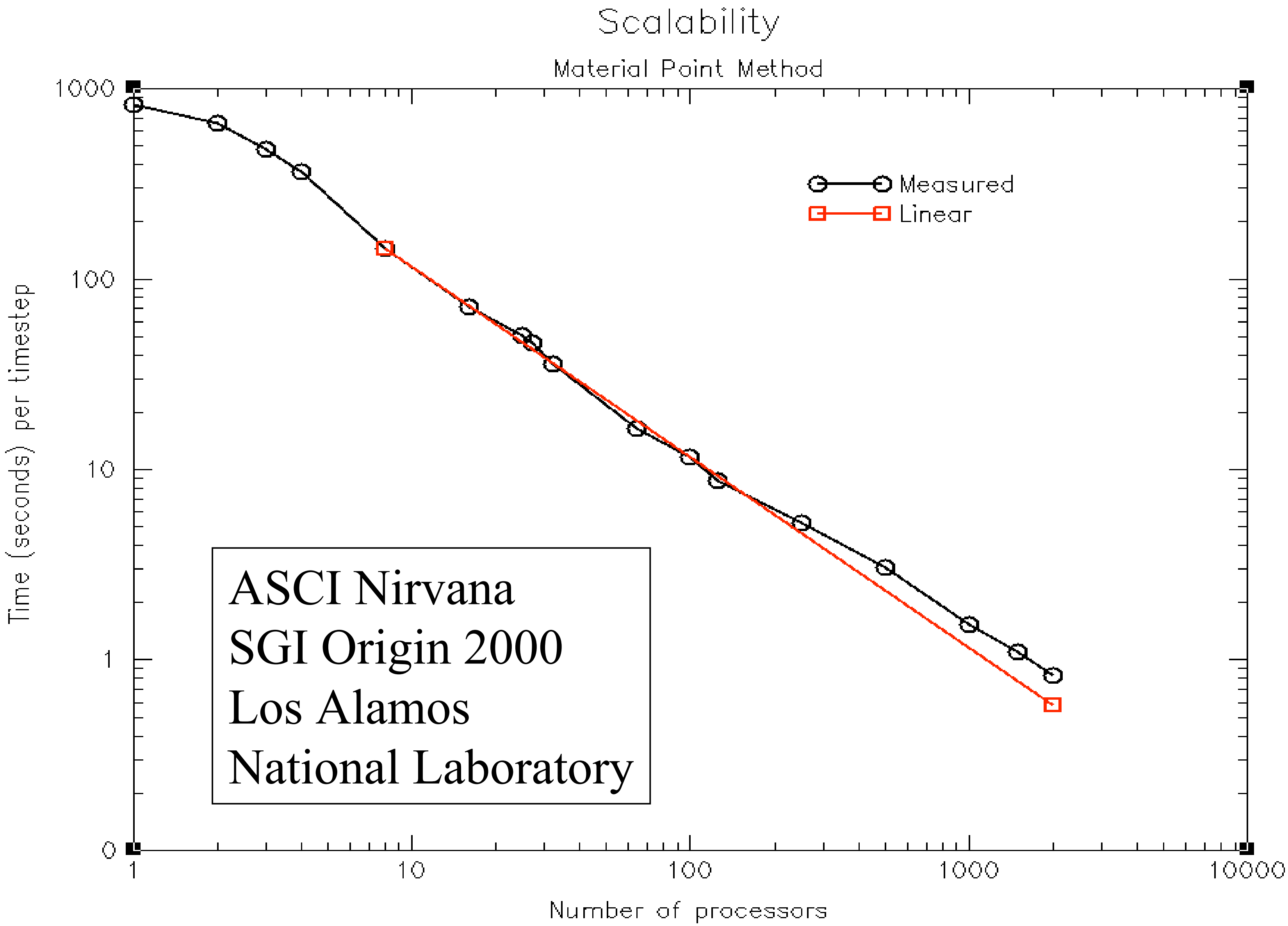
Comparing Uintah Traces for Scalability Analysis



Scaling Performance Optimizations (Past)



Scalability to 2000 Processors (Current)



HYbrid Coordinate Ocean Model (HYCOM)

- Climate, weather and ocean (CWO) based application
  - Primitive equation ocean circulation model (MICOM)
  - Improved vertical coordinate scheme that remains isopycnic in the open, stratified ocean
- Transitions smoothly
  - To *z-level* coordinates in the weakly-stratified upper-ocean mixed layer, to sigma coordinates in shallow water conditions and back to *z-level* coordinates in very shallow water conditions
- User has control over the model domain
  - Generating the forcing field
- Dr. Avi Purkayastha

Getting started with HYCOM

- For generation of the serial or parallel executables
  - Make.com => ../config/$(ARCH)_$(TYPE)
  - Contains all Makefile macro definitions for the preprocessor, fortran, C, parser, and instrumentor compile and link options and associated libraries
- Serial runs
  - 2.00 degree Atlantic Ocean regional grid was used for input, without any change to domain or resolution
- Parallel runs
  - Global ocean (GLBA0.24) was the domain
  - Performance analysis was carried out only for the MPI version of HYCOM

Gprof Profile Data on HYCOM

gprof profile excerpt for serial HYCOM run

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
19.2	312.75	312.75	5400	57.92	62.64	momtum
12.0	507.94	195.19	2700	72.29	139.20	tsadvc
9.9	668.57	160.63	2700	59.49	62.18	cnuity
8.5	807.76	139.19	118800	1.17	1.30	mod_advem
6.3	910.75	102.99				.sqrt (*library*)
5.3	997.17	86.42	140504	0.62	0.62 .	hybgenaj

gprof profile excerpt for parallel HYCOM run

%time	cumulative (seconds)	self seconds	calls	self ms/call	total ms/call	name
21.0	30801.17	30801.17				cors_newpkts (*library*)
16.1	54375.47	23574.30		.		kickpipes (*library*)
14.0	74918.15	20542.68	5760	3566.44	3757.85	momtum
7.9	86522.66	11604.51	5760	2014.67	3764.78	tsadvc
6.5	96048.19	9525.53	191391936	0.05	0.06	mxkppaij
6.2	105110.34	9062.15	5760	1573.29	1603.96	cnuity
5.3	112831.86	7721.52	299520	25.		mod_advem

Gprof Profile Data on HYCOM

- Gprof profile data conclusions
  - Prime candidates for optimization
    - ***momtum***
    - ***tsadvc***
    - ***mx****
    - ***cnuity***
  - mathematical functions
  - also contribute significantly to the run time
  - ***atan*** and ***sqrt***

TAU Profile Analysis of HYCOM

- Exclusive time shows the relative largest time consuming functions
  - Including, surprisingly, MPI_Waitall
- Exclusive time spent can be used as an indicator for measuring efficiency of these functions
  - For example, obtaining MFLOP rates

Tau profile data excerpt with the highest time-consuming functions

%Time	Exclusive msec	Inclusive total msec	#call	#subrtns	Inclusive usec/call	Name
100.0	8:25.60	49:01.37	1	54686.1	2941369933	HYCOM
34.3	16:47.70	16:47.70	54191.4	0	18595	MPI_Waitall
32.1	10:41.98:6	15:43.07	192	113178	4911808	TSADVC
19.7	:11.67	9:40.60	192	31488	3023979	MOMTUM
9.3	2:12.77	4:33.82	192	20601.6	1426132	CNUITY

TAU Profile Analysis of HYCOM

- Low and high end of the time variance spent by the MPI_Waitall call in some of the processors
- Additional investigation is then required from the tracefiles for better understanding overall communication model

Tau profile data excerpt highlighting load imbalance on MPI_Waitall

Proc #	%Time	Exclusive msec	Inclusive total msec	#call	Inclusive usec/call
13	4.7	2:17.33	2:17.33	54182	2535
14	5.6	2:43.63	2:43.63	54182	3020
18	80.6	39:31.07	39:31.07	54229	43723
23	75.3	36:54.28	36:54.28	54182	40867
24	81.1	39:45.41	39:45.41	54229	43988

PAPI profile analysis of HYCOM

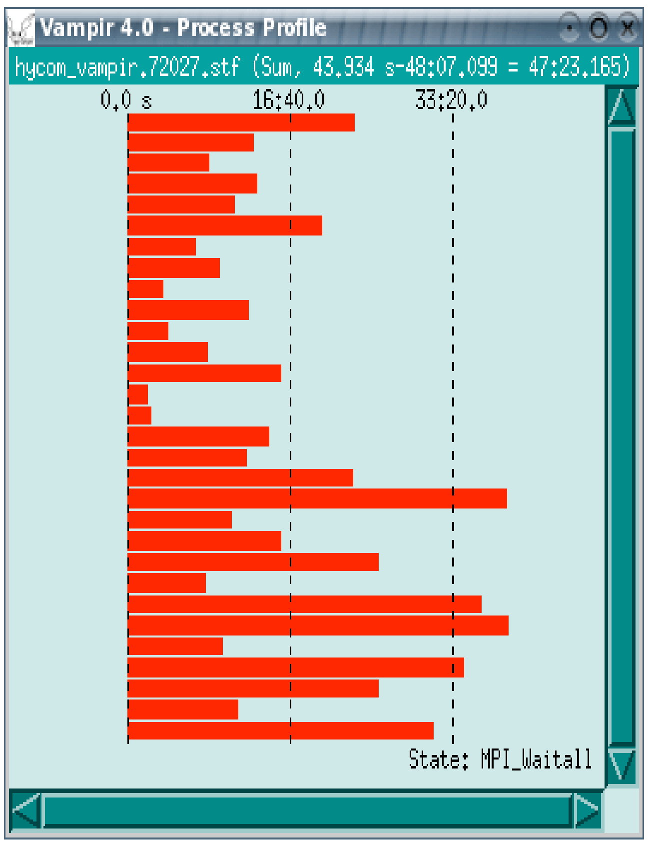
- PAPI profiling (obtained with Tau) exclusive operation count can show performance of individual functions
  - TSADVC
    - 2.07e+11/10:41.98(=642s) = 323 Mflops (~6% of peak)

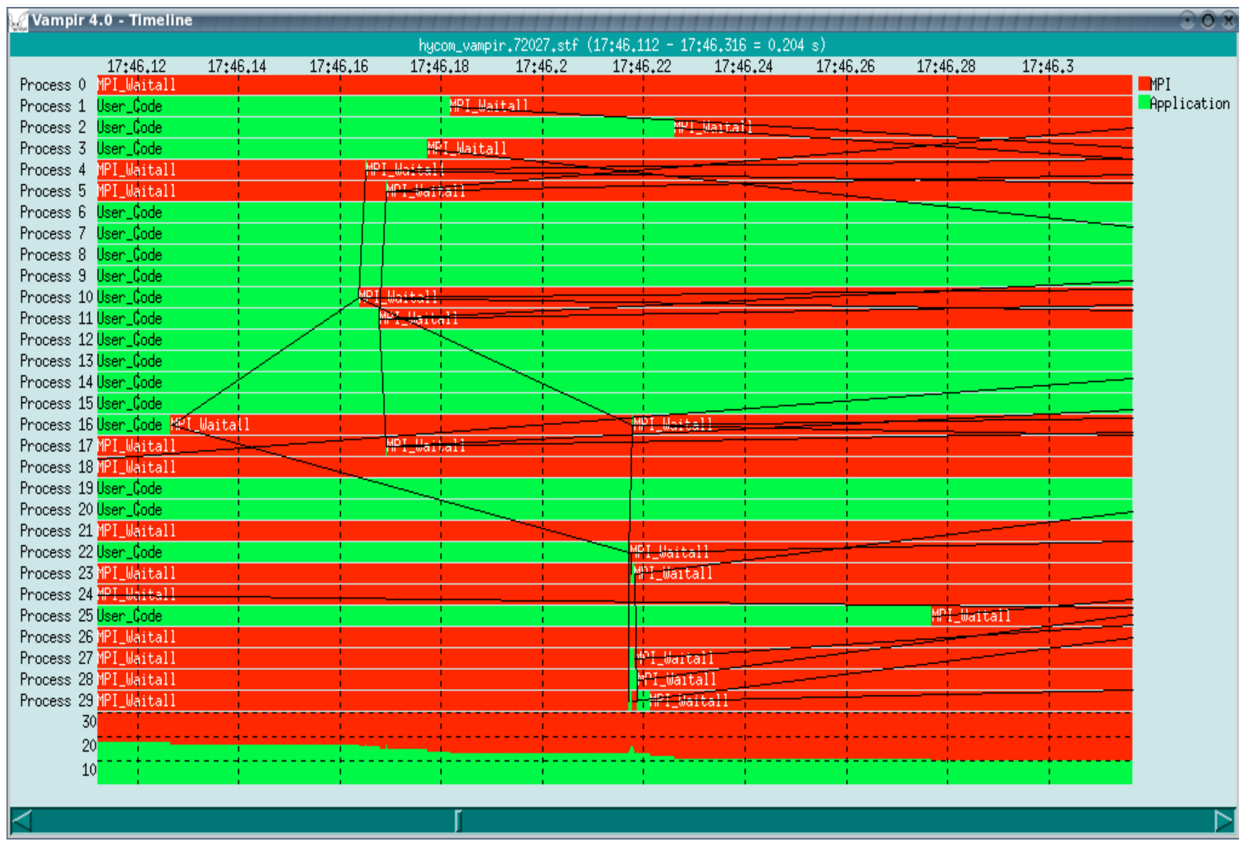
Papi profile data excerpt with the highest time-consuming functions

%Time	Exclusive counts	Inclusive total counts	#call	#subrtns	Inclusive usec/call	Name
100.0	1.56e+11	5.25e+11	1	54686.1	52543238273	HYCOM
39.3	2.07e+11	2.08e+11	192	113178	1085499128	TSADVC
22.2	1.15e+11	1.16e+11	192	31488	606724005	MOMTUM
8.3	4.32e+10	4.37e+10	192	20601.6	227820049	CNUITY
0.5	2.57e+09	2.57e+09	54191.4	0	47497	MPI_Waitall

Vampir Trace analysis of HYCOM

- ***MPI_Waitall*** time per process shows the wide disparity
  - Time spent for blocking call to return => load imbalance
- Small snapshot of global timeline indicates ***MPI_Waitall*** waiting on non-blocking receive operations to complete





Vampir Trace analysis of HYCOM

- This kind of load imbalance is a generic problem for structured-grid ocean models caused by variations in amount of ocean per tile
- HYCOM avoids all calculations over land, so the load imbalance leads to long intervals spent in MPI_Waitall on processors that "own" little ocean
- A different MPI strategy is perhaps necessary to reduce the MPI overhead on those processors with the most computational overhead which is the main contributing factors for those processors waiting for non-blocking receive operations

Air-Vehicles Unstructured flow Solver (AVUS)

- CFD application formerly known as Cobalt60
  - Parallel, implicit Euler/Navier-Stokes 3-D flow solver
    - Second order accurate in space and time
- Solver accepts unstructured meshes composed of a mix of hexahedra, prisms, pyramids, and tetrahedron
- Solver employs a cell-centered, finite-volume method
  - Maintains a compact stencil on an unstructured mesh
- Solver incorporates two one-equation turbulence models
  - Spalart-Allmaras and Baldwin-Barth models
  - Either can be chosen for a flow simulation

Gprof Profile Data on AVUS

- Case performs 20 pseudo-time-steps with 4 Newton sub-iterations per step and 10 block Gauss-Seidel sweeps per Newton iteration

serial

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
30.24	3051.08	3051.08	2501	1.22	1.22	ucm6_
17.62	4828.48	1777.40	2500	0.71	0.72	karl6sc_
10.58	5895.68	1067.20	2500	0.43	0.43	dfdqv6sc_
6.62	6563.80	668.12	2500	0.27	0.27	dfdqi6sc_
5.29	7097.33	533.54	2500	0.21	0.21	lhslusc_
5.19	7621.29	523.96	2500	0.21	0.21	preset6sc_
3.36	7960.23	338.94	2500	0.14	0.14	vflux6_
2.97	8259.75	299.52	2500	0.12	0.12	dfdqt6sc_
2.77	8539.25	279.49	2501	0.11	0.11	riemann_

128 procs

time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
17.64	3144.27	3144.27				smpi_net_lookup
10.97	5099.47	1955.20	320128	6.11	6.11	ucm6_
9.58	6806.52	1707.05	320000	5.33	5.41	karl6sc_
6.05	7884.48	1077.96	320000	3.37	3.37	dfdqv6sc_
5.37	8841.78	957.30	320000	2.99	2.99	dfdqi6sc_
5.28	9782.93	941.15	320000	2.94	2.94	preset6sc_
5.16	10702.34	919.41				MPID_SMP_Check_incoming
4.88	11571.45	869.11				gm_ntoh_u8
3.86	12258.93	687.48				MPID_SendComplete
3.85	12944.93	686.00				gmpi_net_lookup
3.02	13483.33	538.40	320000	1.68	1.68	lhslusc_

Gprof Profile Data on AVUS

- Gprof profile data conclusions
  - Prime candidates for optimization
    - ***karl6sc***
    - ***ucm6***
- Increasing parallelism
  - percentage of utilization of these functions is closer
  - Potential impact of the MPI strategy needs investigation

PAPI profile analysis of AVUS

Papi counter values (millions) for KARL6SC and UCM6 routines

	UCM6	KARL6SC
Total CPU cycles	4630	5220
Total instructions	1772	1495
FP operations	361	264
TLB total misses	2.50	7.10
L1 Data Cache misses	131	104
L2 Data Cache misses	33.3	55
Correct branch-prediction	37.7	6.62
Branch miss-prediction	1.98	0.001

PAPI profile analysis of AVUS

- Entire outer time-step loop in the solver
  - 20 time steps, 4 Newton iterations
  - Instrumented with PAPI functions (via TAU)
  - Measured for floating point performance
    - 204 Mflops ~3.33% of theoretical peak
- Performance is representative of such types of applications with unstructured data-structures
- Detailed PAPI analysis of ***karl6sc*** and ***ucm6*** show the relatively high TLB data misses
  - More detailed loop analysis were then performed with PAPI to isolate the bottlenecks causing these TLB misses

TAU Performance Analysis of AVUS

- Mean profile of the original AVUS source
  - Running on 16 procs, using 44 matrix solve sweeps
  - ***karl6sc*** dominates the calculations
  - ***ucm6*** and ***MPI_Ssend*** take almost the same time

FUNCTION SUMMARY (mean):

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive usec/call	Name
59.0	1:00.129	1:13.608	180	15480	408939	KARL6SC
11.1	13,906	13,906	181	0	76832	UCM6
10.9	13,648	13,648	34995.6	0	390	MPI _Ssend()
5.2	6,498	6,498	180	0	36104	DFDQV6SC
4.5	5,626	5,626	180	0	31256	DFDQI6SC
3.3	4,144	4,144	180	0	23028	PRESET6SC
2.1	2,677	2,677	8764.5	0	306	MPI_Waitall()
1.9	2,389	2,389	180	0	13272	LHSLUSC
1.7	2,163	2,163	181	0	11955	RIEMANN
1.7	2,079	2,079	180	0	11553	DFDQT6SC
1.3	1,662	1,662	181	0	9184	GRAD6
1.1	1,358	1,358	180	0	7546	VFLUX6
76.5	1,147	1:35.507	180	1100	530597	INTEGR86
0.9	1,145	1,145	164.25	164.25	6977	MPI_Bcast()

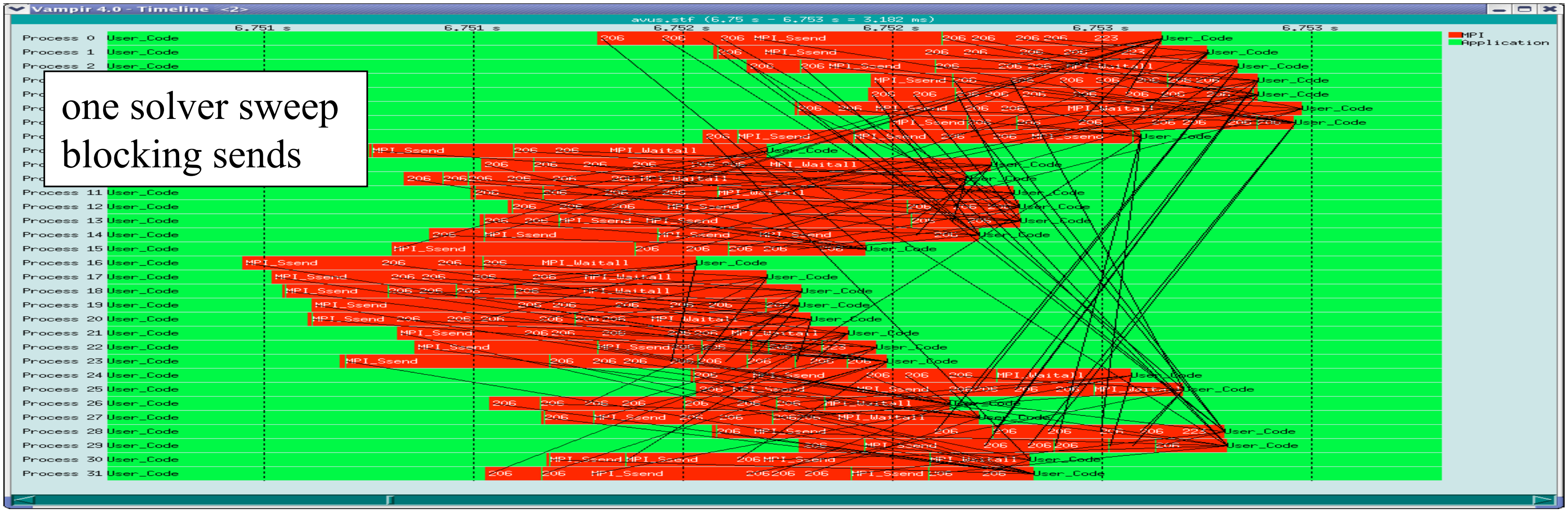
Tau performance analysis of AVUS

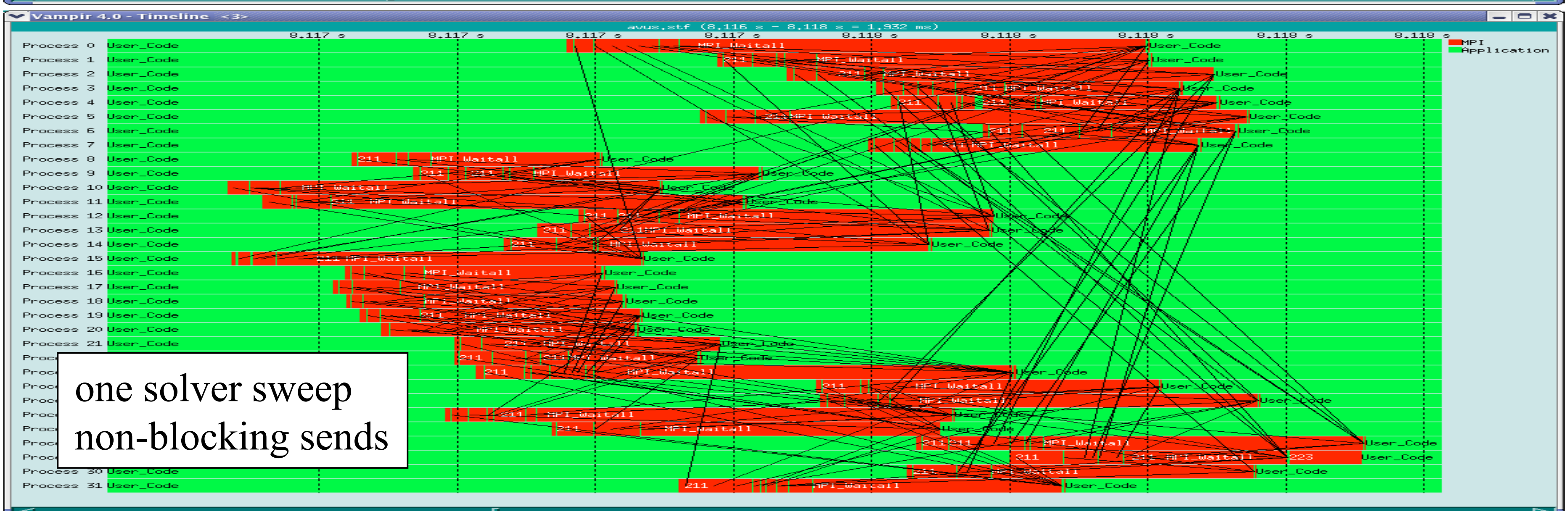
- Mean profile of the original AVUS source
  - 16 procs using convergence checks (44 solver sweeps)
  - With ***ucm6*** improvements and ***MPI_Ssend*** replacements

FUNCTION SUMMARY (mean):

%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive usec/call	Name
54.2	50,402	1:03.235	180	18009	351308	KARL6SC
11.0	12,823	12,823	181	0	70848	UCM6
6.7	7,835	7,835	13632.5	0	575	MPI_Waitall()
6.2	7,193	7,202	6385	6385	1128	MPI_Allreduce()
5.6	6,534	6,534	180	0	36302	DFDQV6SC
4.8	5,574	5,574	180	0	30969	DFDQI6SC
3.4	4,023	4,023	180	0	22354	PRESET6SC
2.4	2,831	2,832	1	39	2832007	MPI_Init()
2.1	2,417	2,417	180	0	13433	LHSLUSC
1.0	1,199	1,199	28519.8	0	42	MPI_Isend()

Vampir Trace Analysis for AVUS (32 processor)

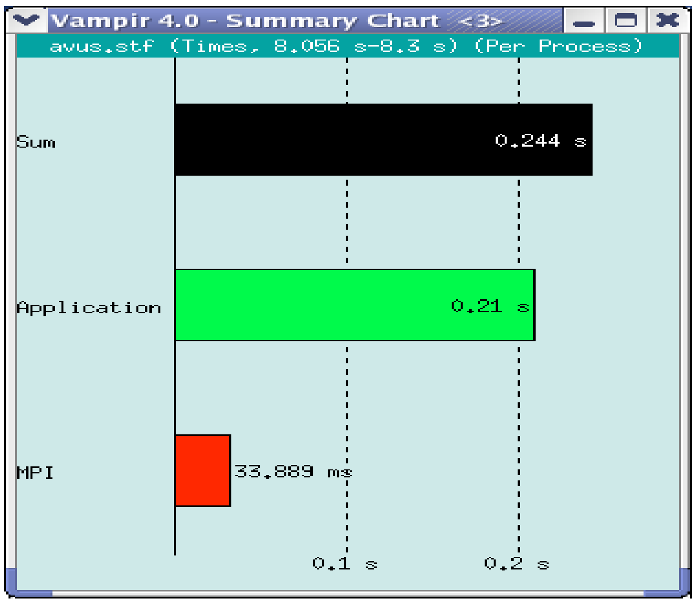




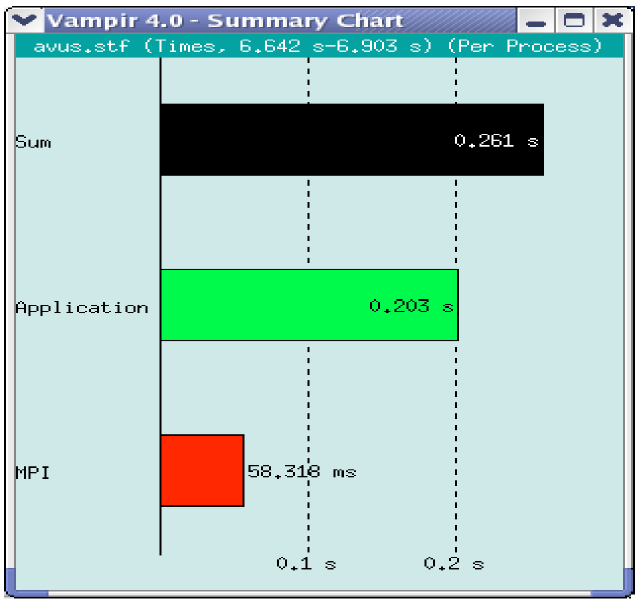
Vampir trace analysis for AVUS

- From timeline and summary graphs
  - Effect of replacing ***MPI_Ssends***
  - Use ***MPI_Isends*** and ***MPI_Wait***

Process summary -- 32 procs w/non-blocking sends

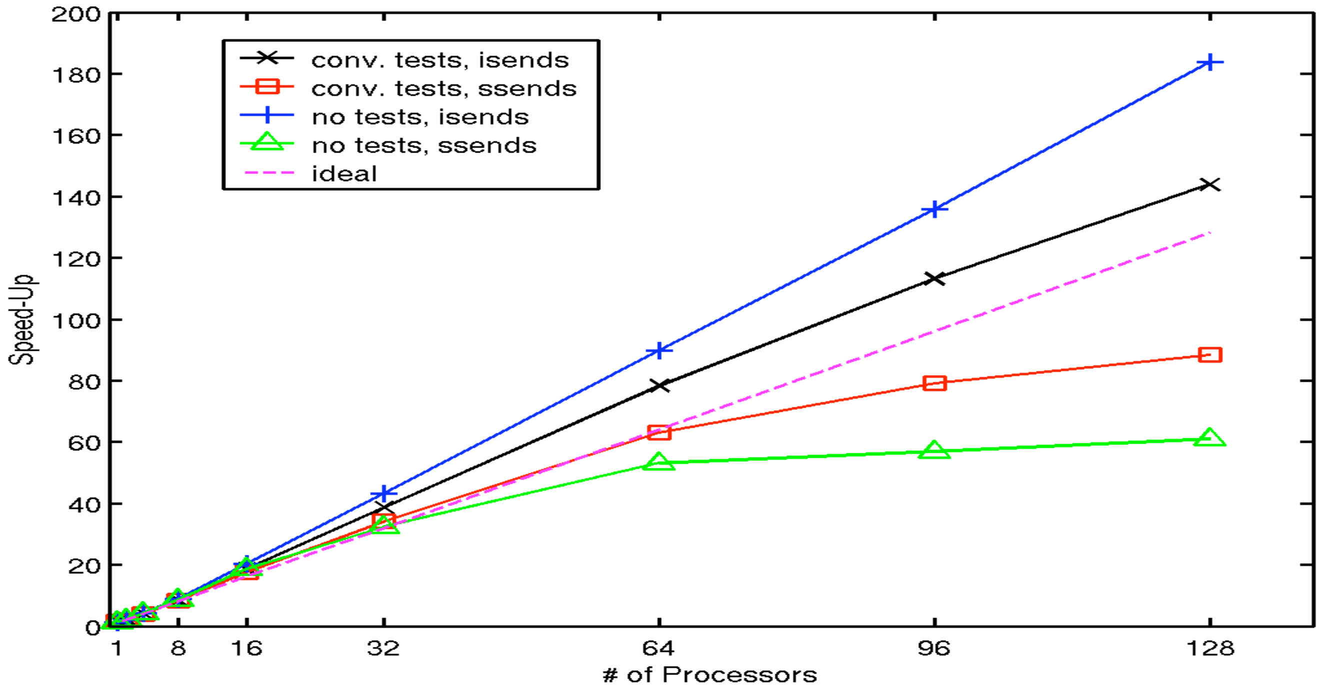


Process summary -- 32 procs w/blocking sends



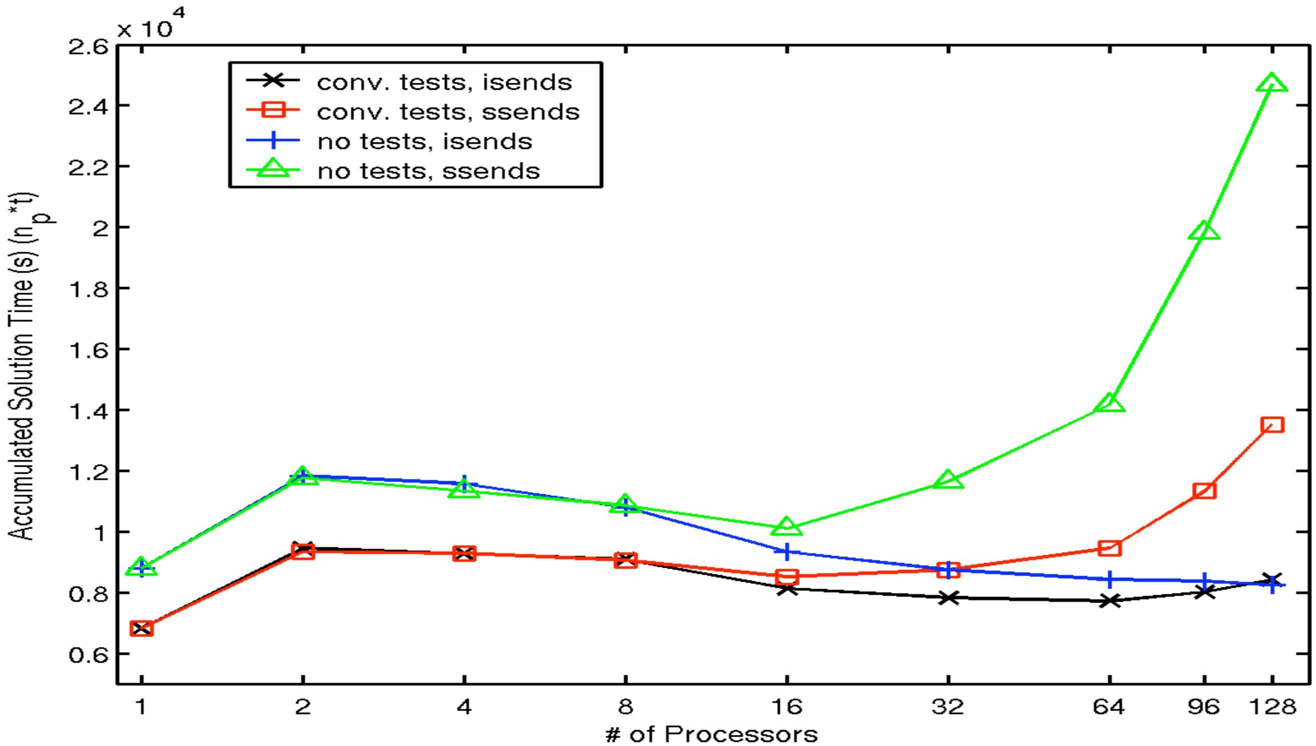
AVUS Scalability results

- Final scalability results
  - Original AVUS source with/without convergence tests
  - Blocking vs. non-blocking communications



AVUS Scalability Results

- Final accumulated solution times
  - Original AVUS source with/without convergence tests
  - Blocking vs. non-blocking communications



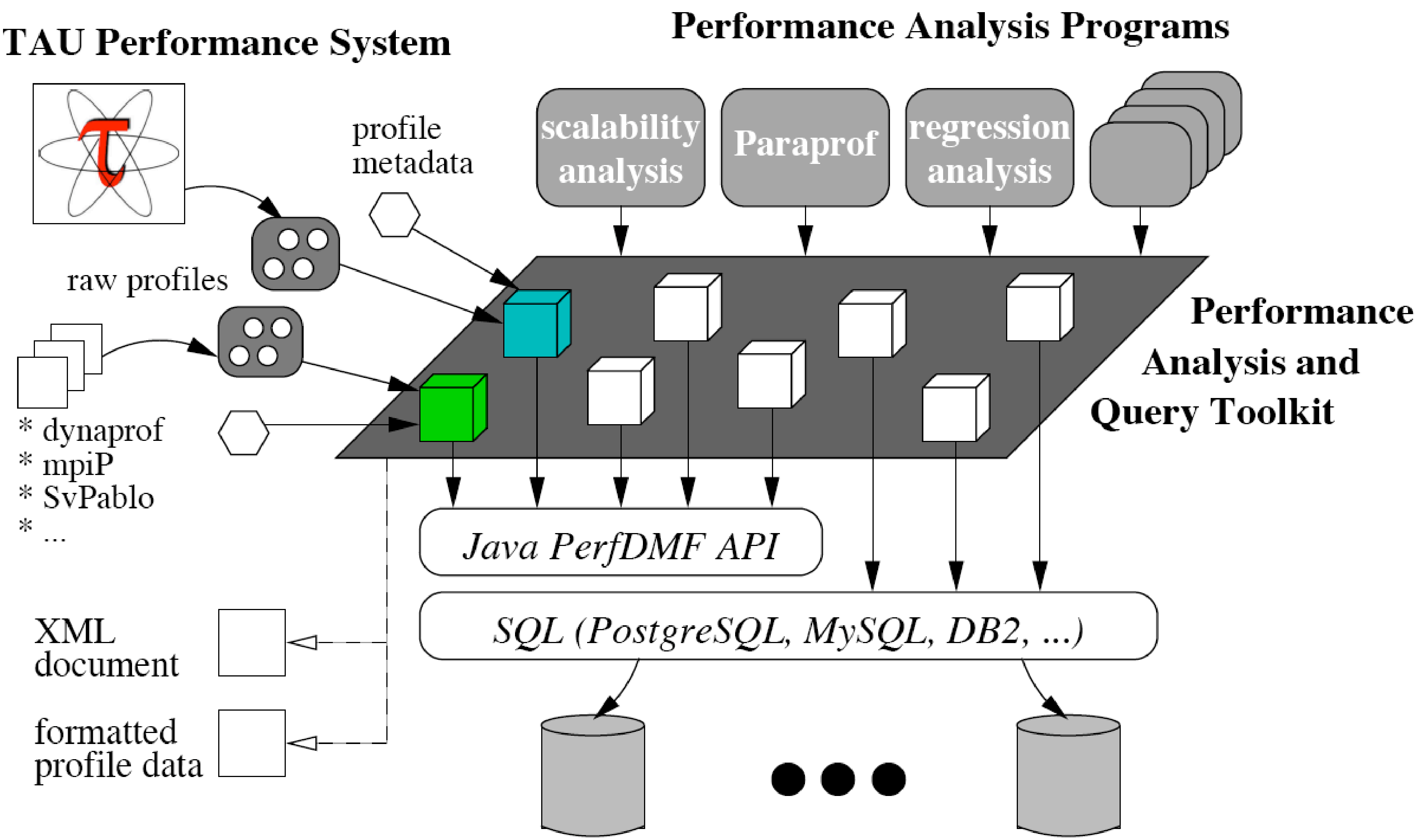
Performance Data Management

- Performance evaluation of parallel programs and systems requires analysis of data from multiple experiments
- Little support exists for storing and evaluating datasets from a variety of experimentation scenarios
- Need open performance data management technology that can provide a common, reusable foundation for performance results storage, access and sharing
- Provide standard solutions for how to represent parallel performance data
- *Performance Data Management Framework (**PerfDMF**)*

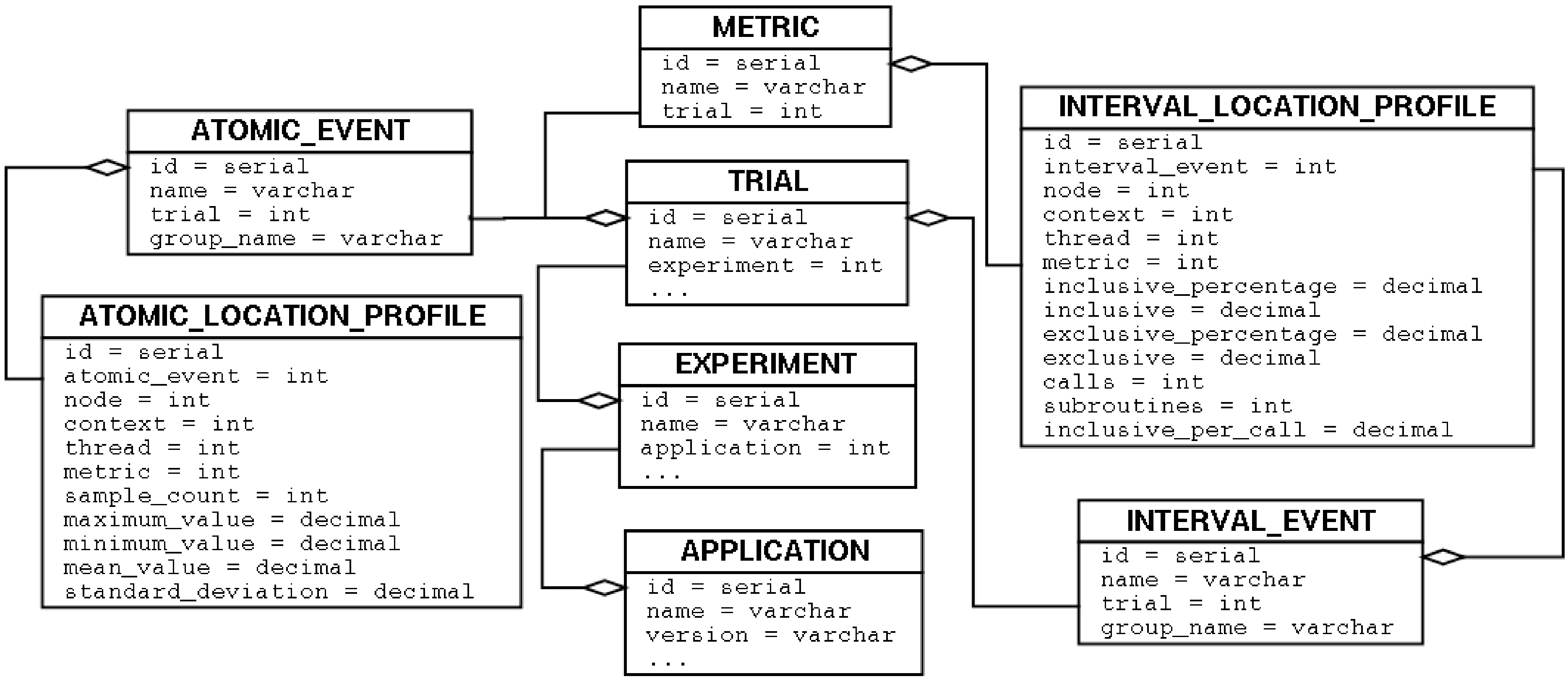
PerfDMF Objectives

- Import/export of data from/to parallel profiling tools
- Handle large-scale profile data and large numbers of experiments
- Provide a robust profile data management system
  - Portable across user environments
  - Easily reused in the performance tool implementations
  - Able to evolve to accommodate new performance data
- Support abstract profile query and analysis API that offers an alternative DBMS programming interface
- Allow for extension and customization in the performance data schema and analysis API

TAU Performance Database Architecture



PerfDMF Data Schema



PerfDMF Loading Tools

- Configuration Utility / Schema loader
  - Configures database connection settings, loads database schema if necessary
- Application Creator / Loader
  - Loads application metadata from XML file, or creates an application with empty metadata
- Experiment Creator / Loader
  - Loads experiment metadata from XML file, or creates an experiment with empty metadata
- Trial Loader
  - Loads parallel profile data from several supported formats
  - *TAU*, *dynaprof*, *mpiP*, *HPMToolkit*, *gprof*, *psrun*, …

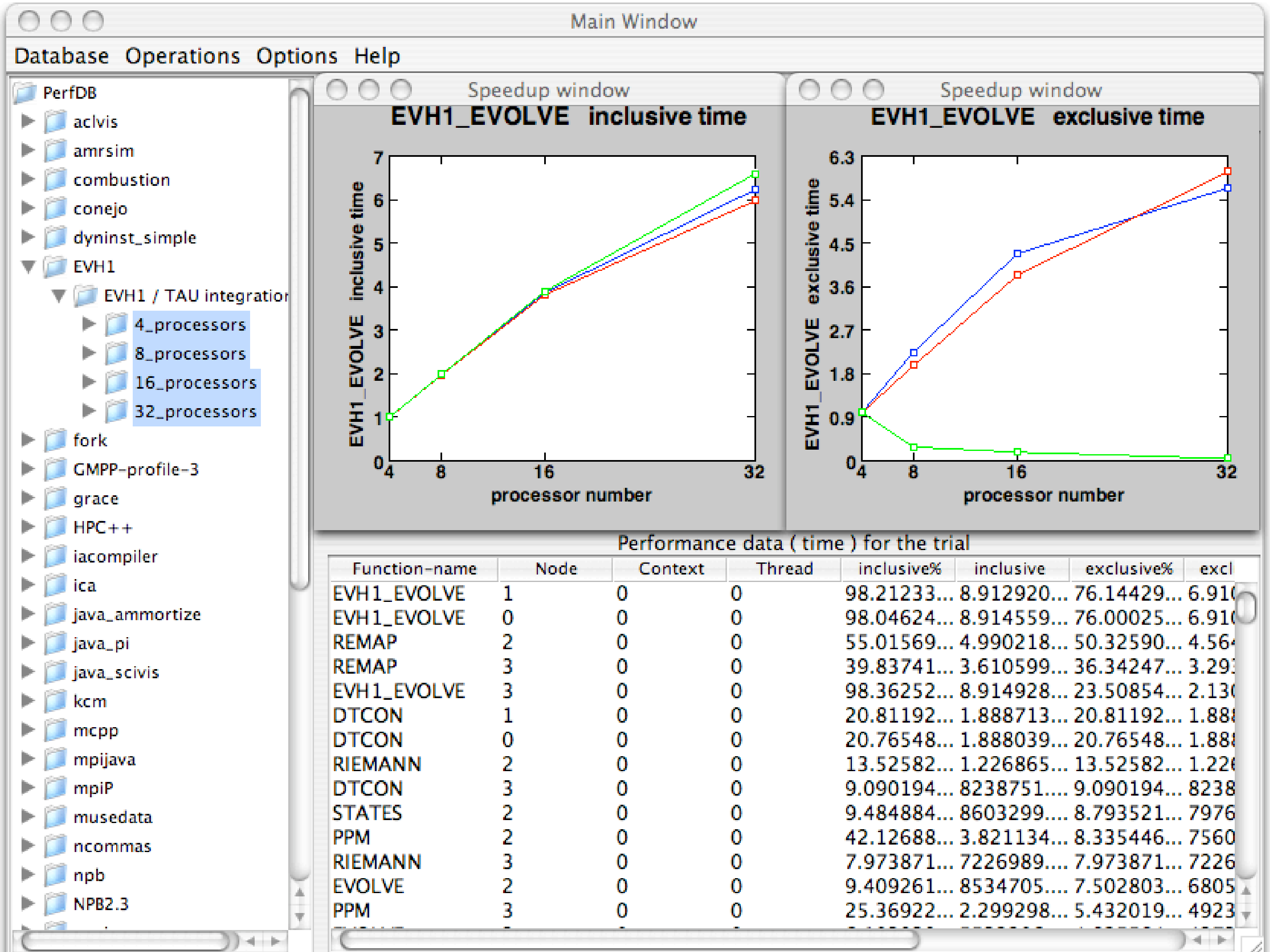
Query / Analysis API

- Java 1.4 API for querying database
- Paradigm
  - Select object / set filter
  - Get list of sub-objects
  - Example:

```
session.setNode(0);
session.setContext(0);
session.setThread(0);
// get the interval events from compute node 0, process 0, thread 0
DataSessionIterator events = session.getIntervalEvents();
while (events.hasNext()) {
        // . . .
}
```

Examples: ParaProf Integration

Example: Profile Viewer



Software Engineering of PerfDMF

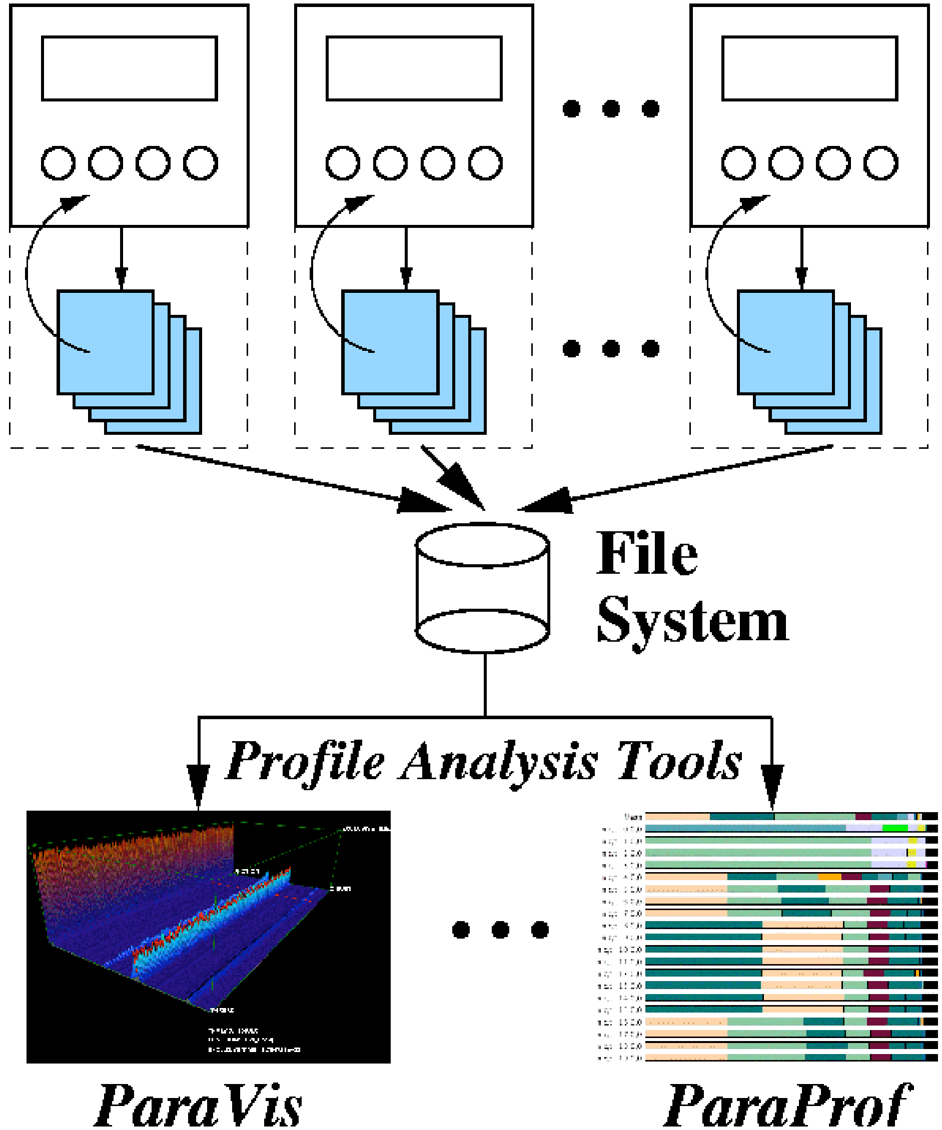
- Java 1.4 - runs anywhere there is a JVM
- JDBC connection allows for use of any supported DBMS
  - Tested with PostgreSQL, MySQL, DB2
- Generic data schema supports different profile formats
  - *TAU*, *dynaprof*, *mpiP*, *HPMToolkit*, *gprof*, *psrun*, …

PerfDMF Future Work

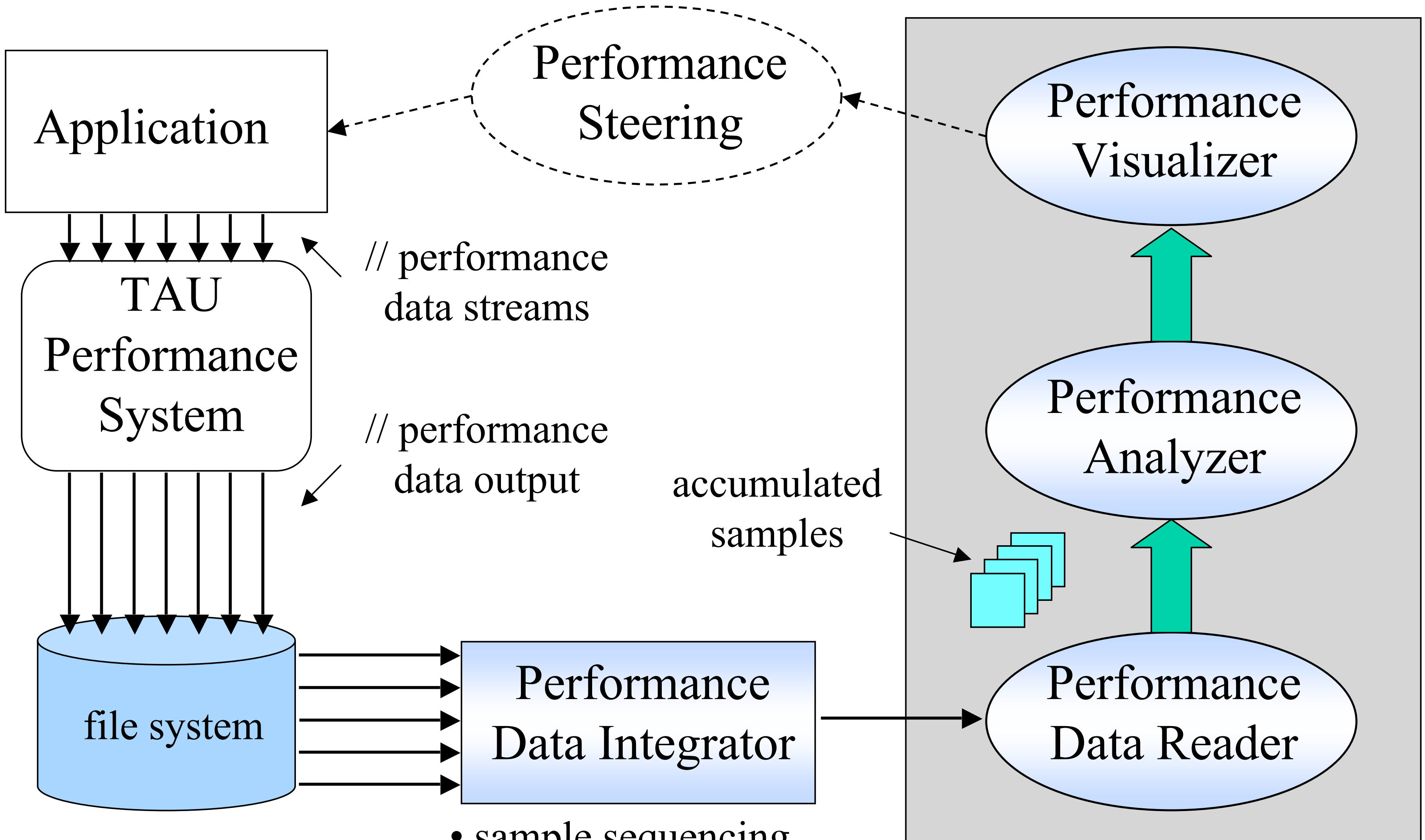
- Short term
  - Integration with CUBE
  - Integration with PPerfDB/PPerfXchange
  - Support more profile formats
  - Application of data mining operations on large parallel datasets (over 1000 threads of execution) .
- Long term
  - Performance profile data and analysis servers
  - Shared performance repositories

Online Profile Measurement and Analysis in TAU

- Standard TAU profiling
  - Per node/context/thread
- Profile "dump" routine
  - Context-level
  - Profile file per each thread in context
  - Appends to profile file
  - Selective event dumping
- Analysis tools access files through shared file system
- Application-level profile "access" routine

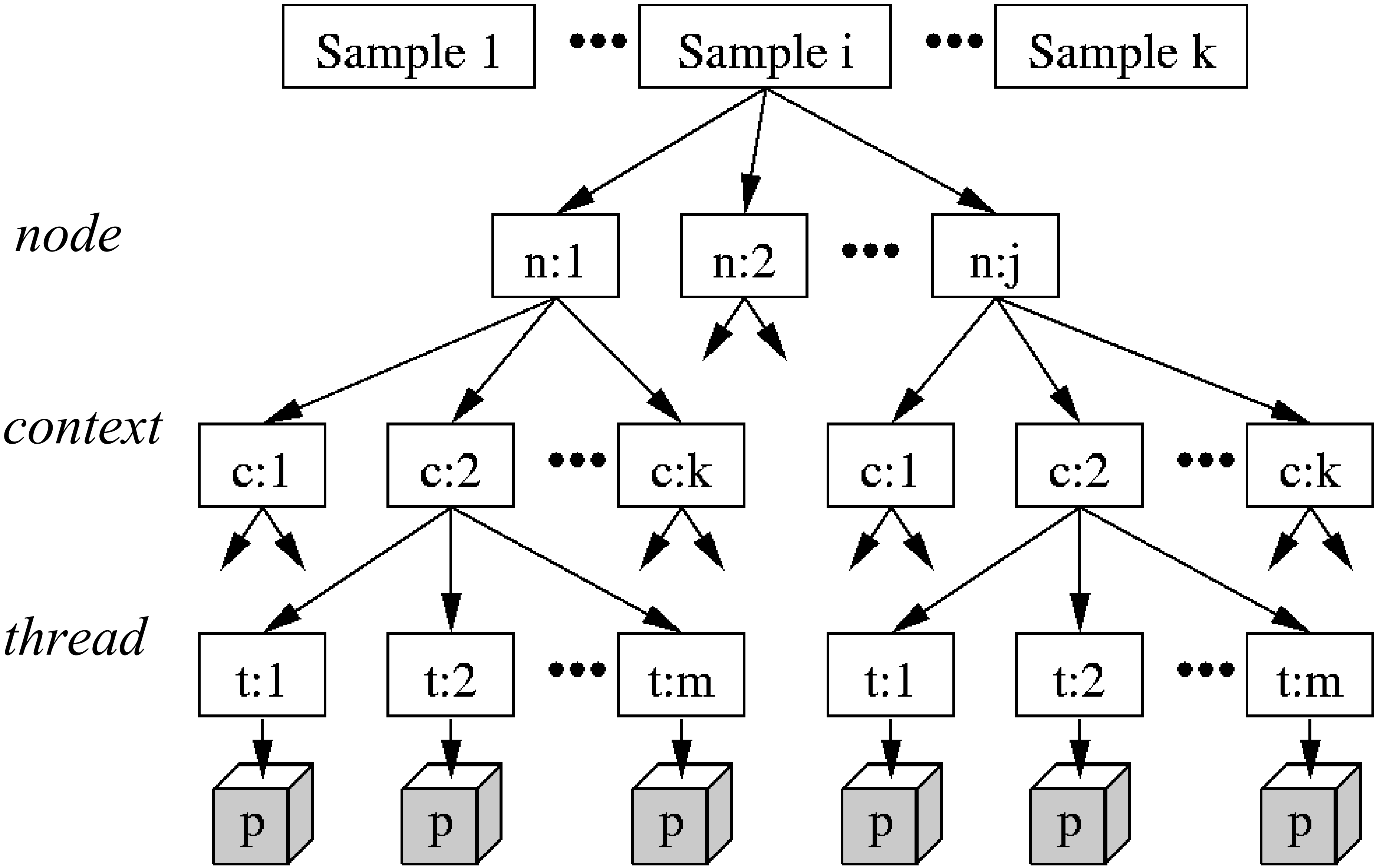


Online Performance Analysis and Visualization

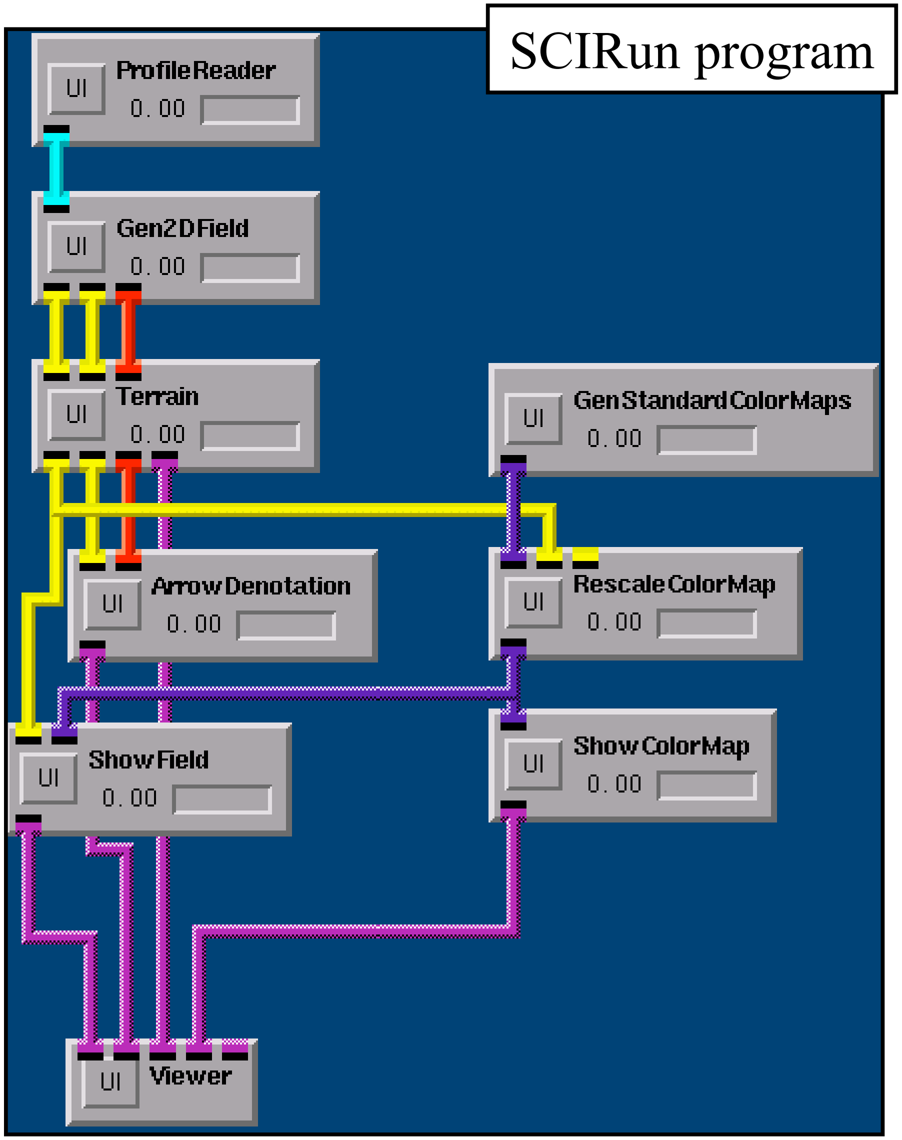


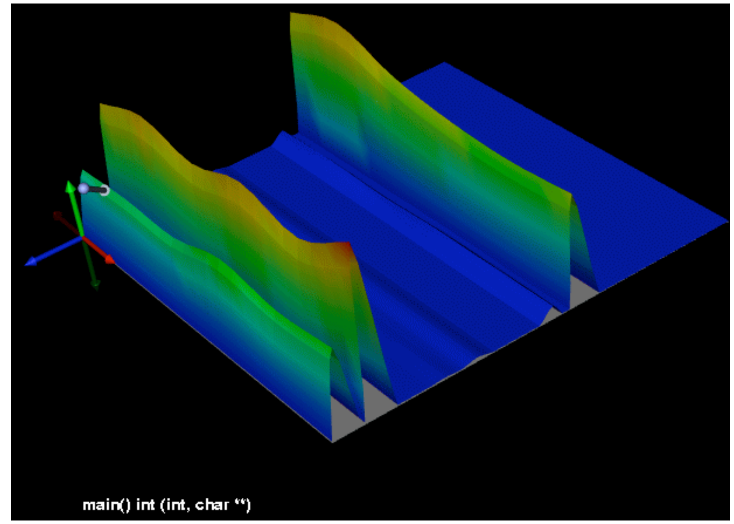
- sample sequencing
- reader synchronization

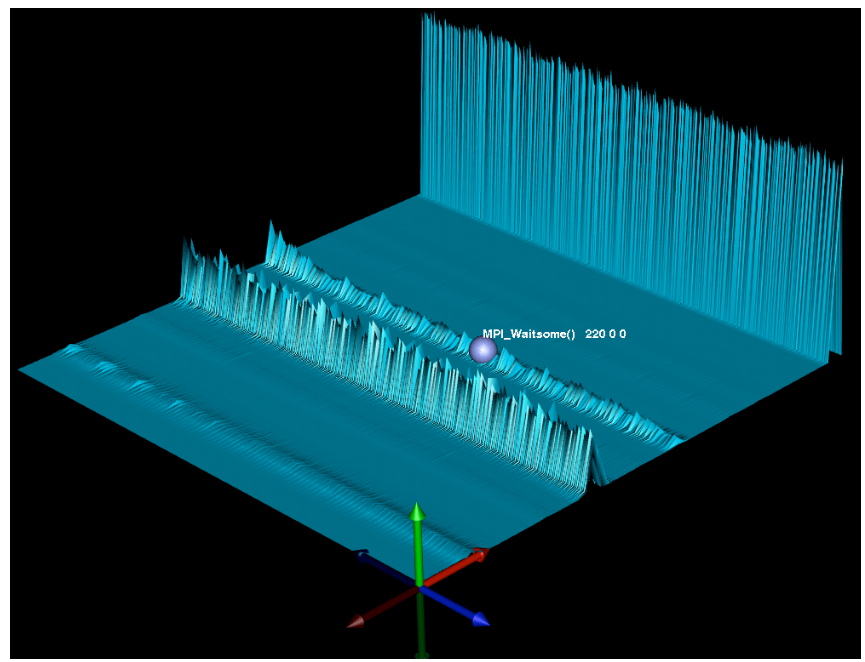
Profile Sample Data Structure



Performance Analysis/Visualization in SCIRun

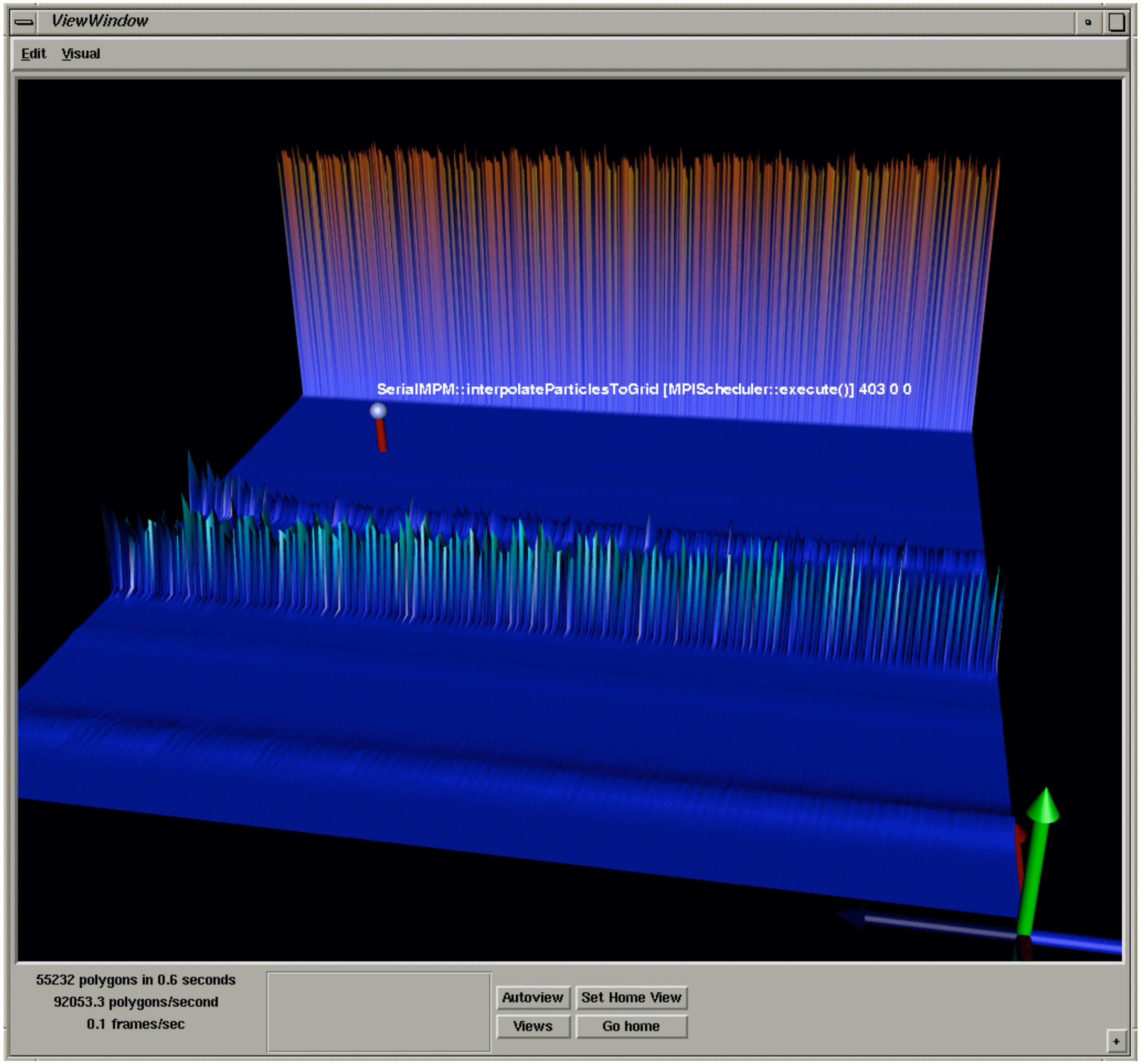




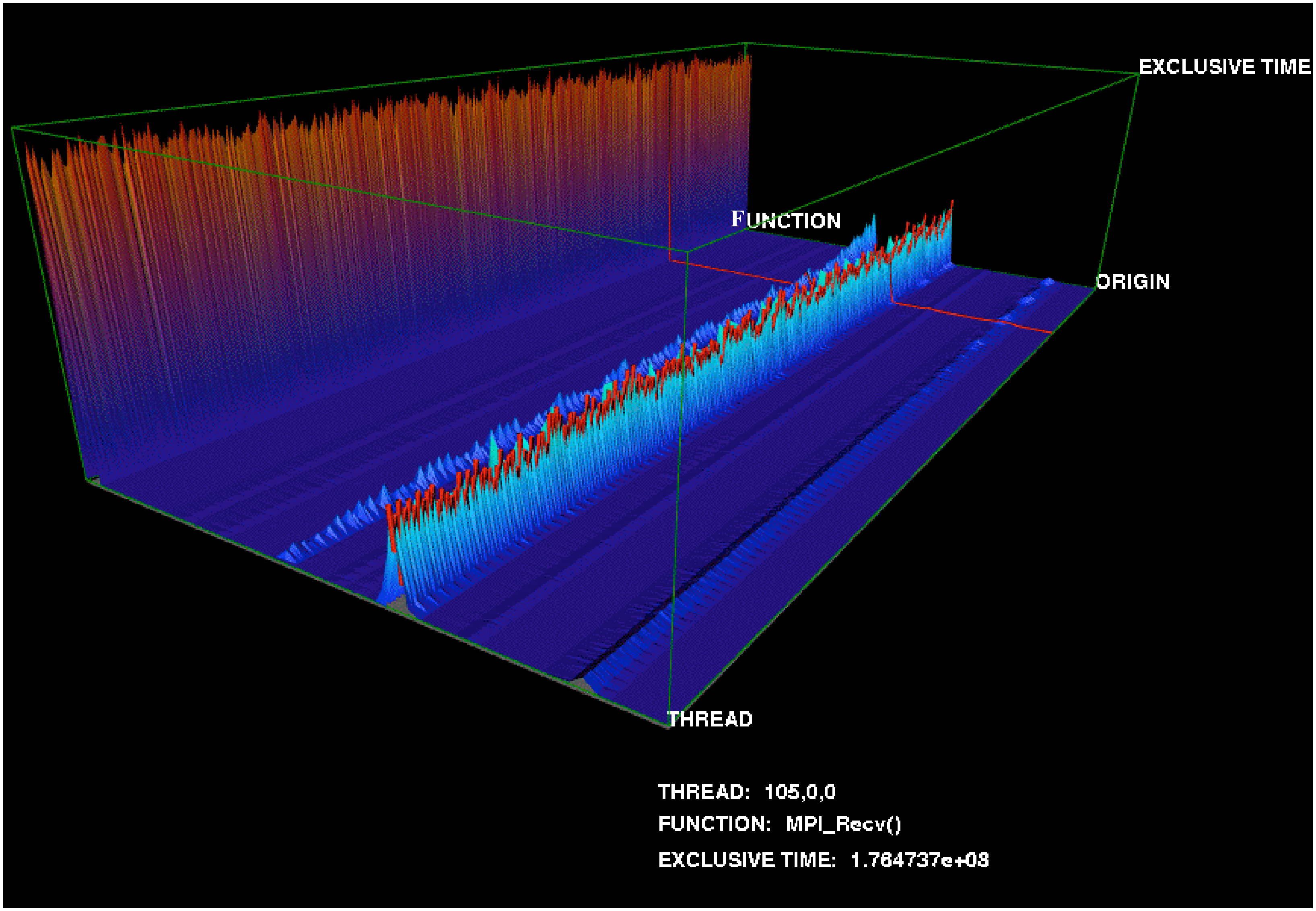


Uintah Computational Framework (UCF)

- University of Utah
- UCF analysis
  - Scheduling
  - MPI library
  - Components
- 500 processes
- Use for online and offline visualization
- Apply SCIRun steering

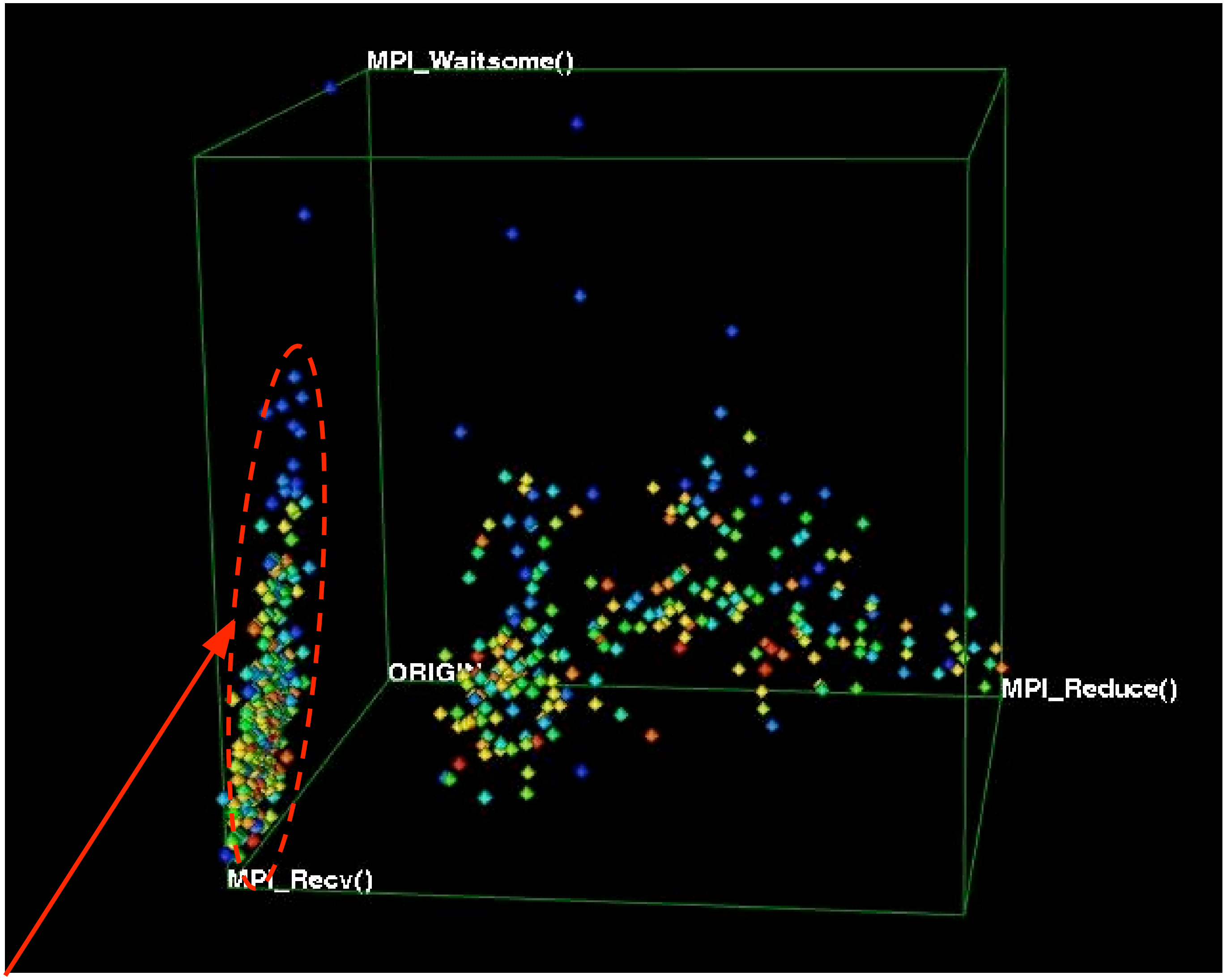


“Terrain” Performance Visualization



Scatterplot Displays

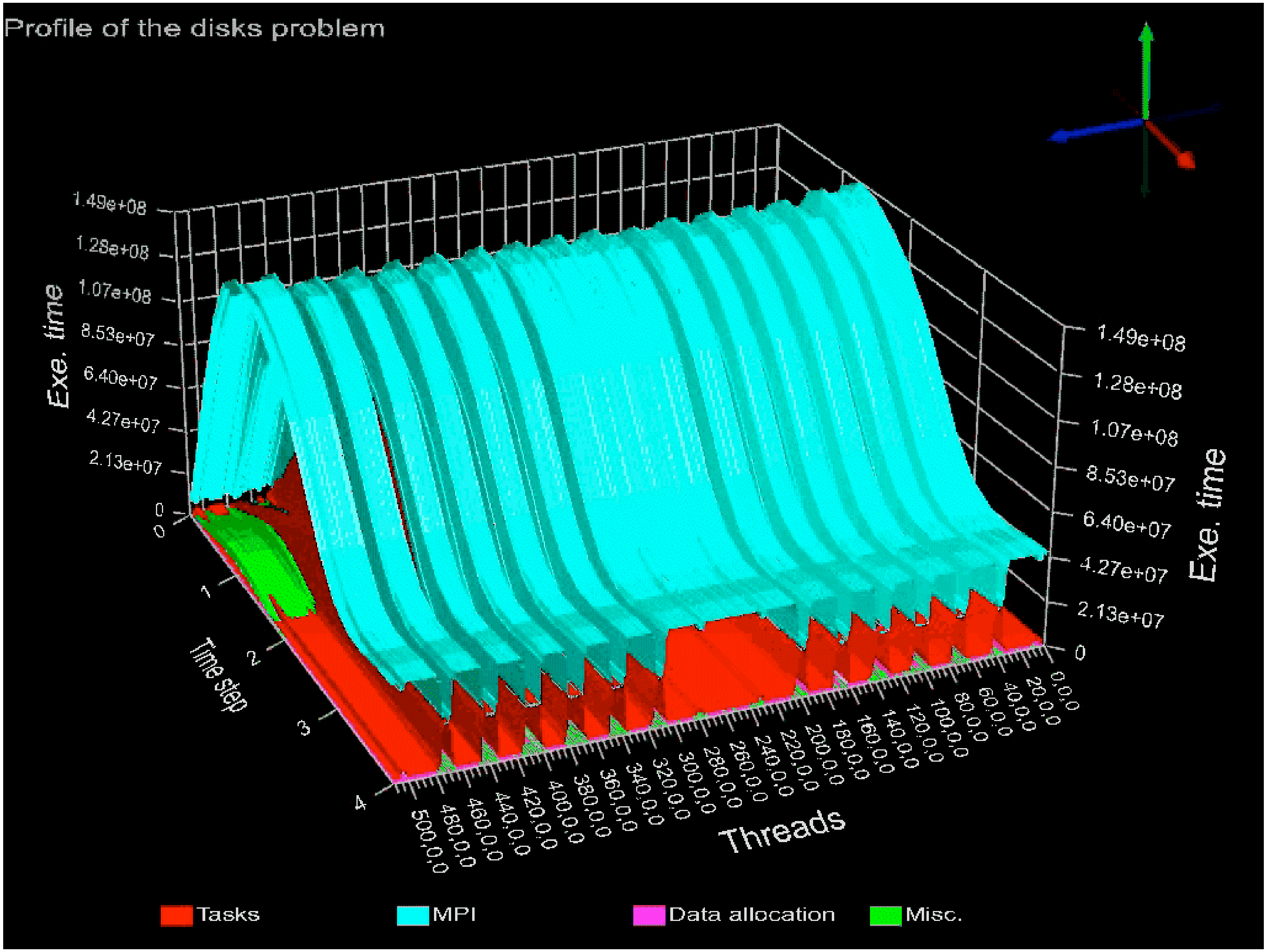
- Each point coordinate determined by three values:
  - *MPI_Reduce*
  - *MPI_Recv*
  - *MPI_Waitsome*
- Min/Max value range
- Effective for cluster analysis
  - Relation between *MPI_Recv* and *MPI_Waitsome*



Online Unitah Performance Profiling

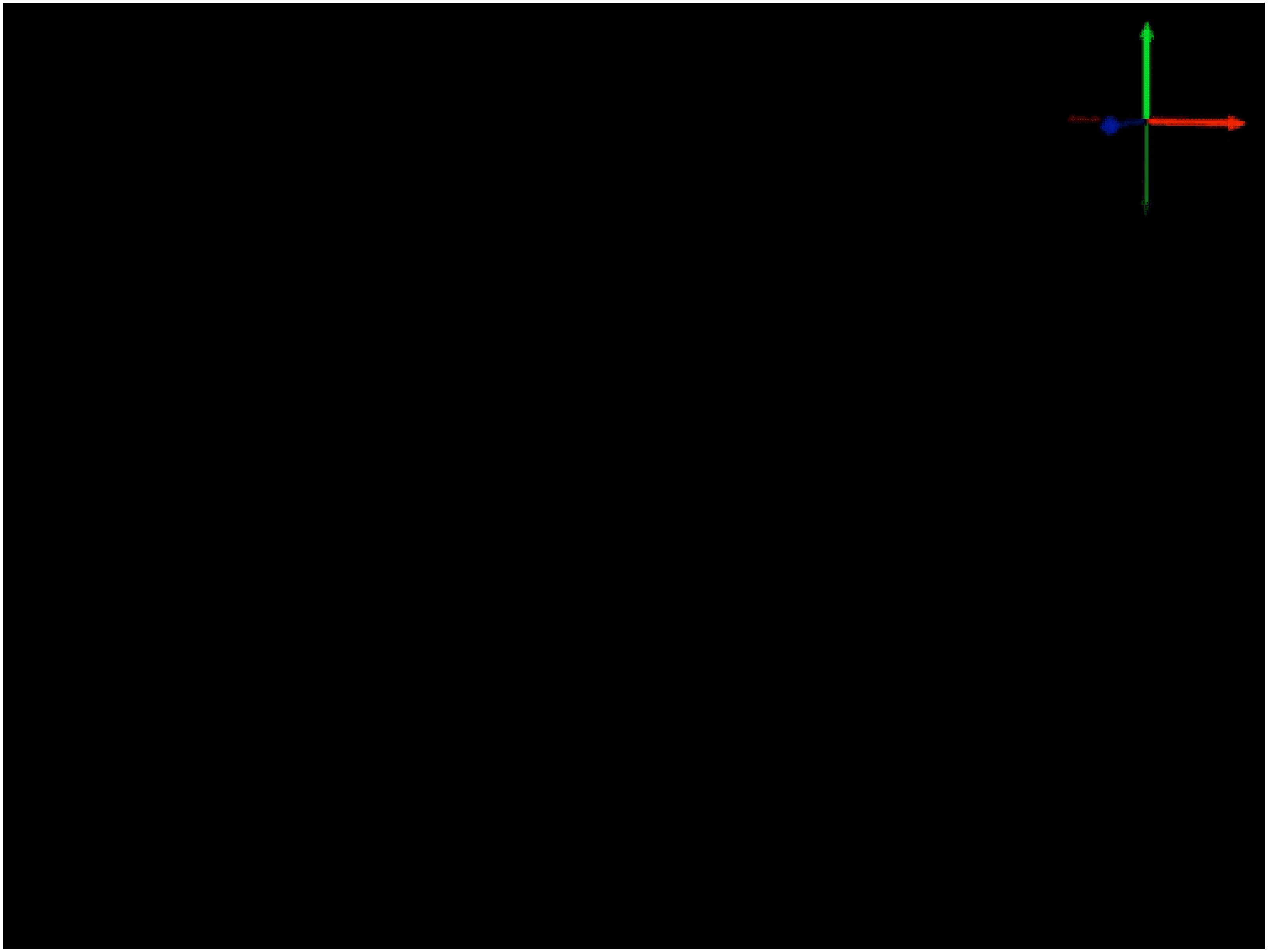
- Demonstration of online profiling capability
- Colliding elastic disks
  - Test material point method (MPM) code
  - Executed on 512 processors ASCI Blue Pacific at LLNL
- Example 1 (Terrain visualization)
  - Exclusive execution time across event groups
  - Multiple time steps
- Example 2 (Bargraph visualization)
  - MPI execution time and performance mapping
- Example 3 (Domain visualization)
  - Task time allocation to "patches"

Example 1 (Event Groups)



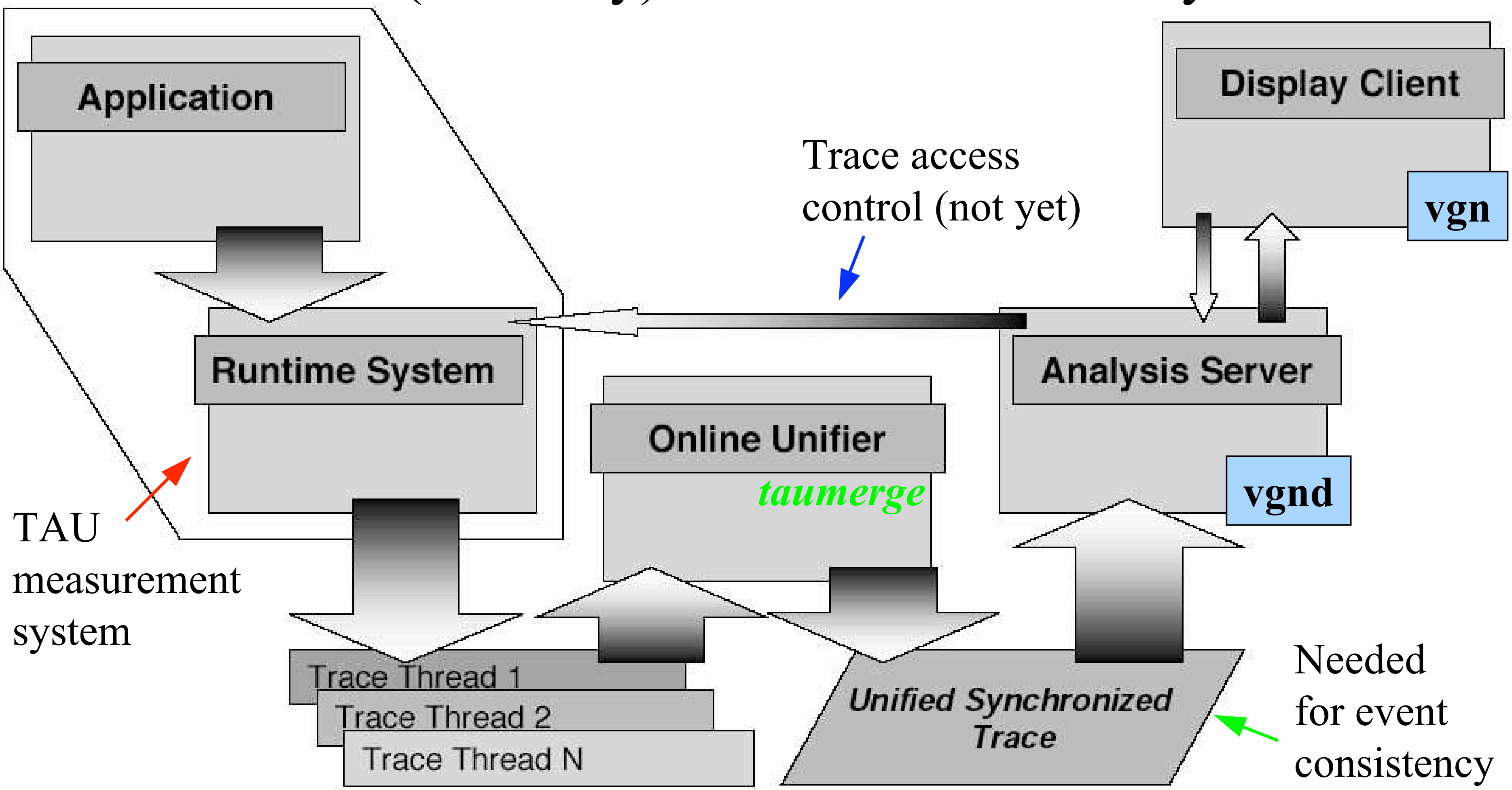
Example 2 (MPI Performance)

Example 3 (Domain-Specific Visualization)



Online Trace Analysis with TAU and VNG

- TAU measurement of application to generate traces
- Write traces (currently) to NFS files and unify



Integrated Performance Evaluation Environment