What it Takes to Assign Blame

Nick Rutar
Jeffrey K. Hollingsworth

University of Maryland

Parallel Framework Mapping

- Traditional profiling represented as
  - Functions, Basic Blocks, Statement
- Frameworks have intuitive abstractions
  - Direct ties with mathematical terms
  - PETSc, Cactus, POOMA, GrACE
- Map profiling information to variables
  - Maps to abstractions in case of frameworks
  - Also can be used for standard programs
    - Map Structs, Classes, Arrays, Scalars

Example PETSC Program*

* - $PETSC_DIR/src/ksp/ksp/examples/tutorials/ex23.c

50% cache misses → Vec

30% MPI operations → Mat

40% run time → PC

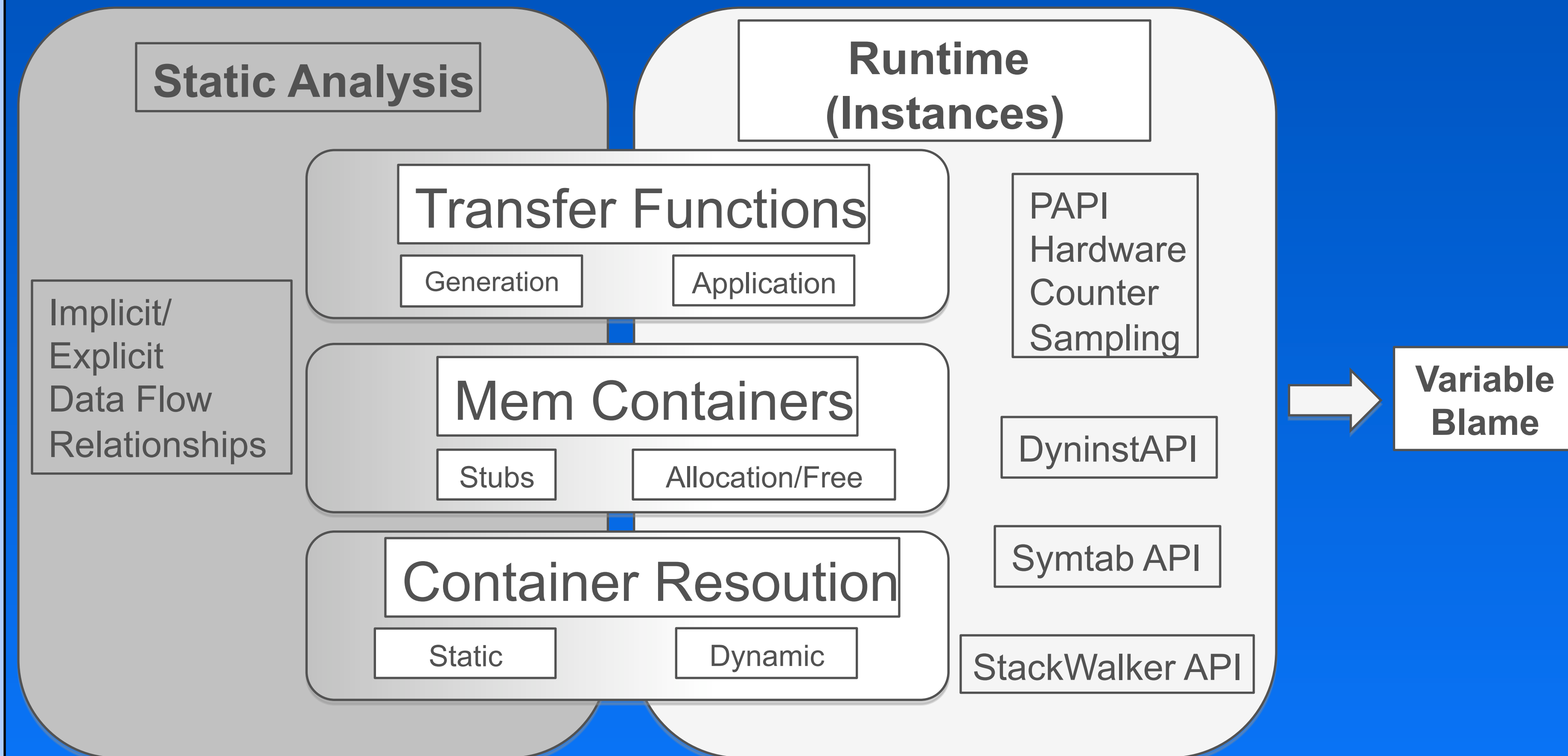
```
int main(int argc,char **args) {
 Vec     x, /* approx solution */
          b, /* right hand side */
          u;  /* exact solution*/
 Mat     A;         /* linear system matrix */
 KSP    ksp;       /* linear solver context */
 PC      pc;         /* preconditioner context */
VecCreate(PETSC_COMM_WORLD,&x);
VecDuplicate(x,&b);
VecDuplicate(x,&u);
MatCreate(PETSC_COMM_WORLD,&A);
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
/* Set exact solution */
 VecSet(u,one);
 MatMult(A,u,b);
 /* Create linear solver context */
 KSPCreate(PETSC_COMM_WORLD,&ksp);
 KSPGetPC(ksp,&pc);
 PCSetType(pc,PCJACOBI);
 /* Solve linear system */
 ierr = KSPSolve(ksp,b,x); }
```


Dyn inst

Variable "Blame"

- Record writes in a function
- Build association tree of writes from ground up
- Use transfer function to filter information up
  - Up the call stack
  - Aggregate over distributed nodes
- Eventually reach high level abstractions
  - Example: Matrix abstraction
    - Allocated storage for actual data
      - Sparse or Dense
    - Storage for bookkeeping
- Augments traditional profiling approaches

Blame Calculation Components



Preliminary Experimental Results

- Chose programs with similar properties to those found in parallel frameworks
- Blame metric is number of cycles
- For each sampling point (instance)
  - Instance gets blamed for set number of cycles
  - Variable that instance maps up to gets blame

FFP_SPARSE

- C++ program that solves Poisson's Equation
  - Approximately 6,700 lines of code & 63 Functions
- Non-parallel program
- Uses Sparse Matrices
  - No specific data structure for representation
  - Composite of primitive pointers declared in 'main'
- Recorded 101 samples from program run

FFP_SPARSE Results

Name	Type	Description	Direct	Blame (%)
node_u	double *	Solution vector	0	35 (34.7)
a	double *	Coefficient matrix	0	24.5 (24.3)
ia	int *	Non-zero row indices of a	1	5 (5.0)
ja	int *	Non-zero column indices of a	1	5 (5.0)
element_neighbor	int *	Estimate of non-zeroes	0	10 (9.9)
node_boundary	bool *	Bool vector for boundary	0	9 (8.9)
f	double *	Right hand side of vector	0	3.5 (3.5)
Other	-		99	9 (8.9)
Total	-		101	101 (100)

HPL

- C program that solves a linear system
  - Utilizes MPI and BLAS
  - Has wrappers for functions from both libraries
  - Operations done on dense matrices
  - Approximately 18,000 lines of code
  - 149 source files
- 32 Red Hat nodes connected via Myrinet
  - OpenMPI 1.2.8
  - Range of 149-159 samples over the nodes

HPL Results

Name	Type	Blame over 32 Nodes: Mean (Total %)	Blame over 32 Nodes: Node St. Dev.
All Instances	-	154.7(100)	2.7
main			
grid	HPL_T_grid	2.2(1.4)	0.4
main→HPL_pdtest			
mat	HPL_T_pmat	139.3(90.0)	2.8
Anorm1	double	1.4(0.9)	0.8
AnormI	double	1.1(0.7)	1.0
XnormI	double	0.5(0.3)	0.7
Xnorm1	double	0.2(0.1)	0.4
main→HPL_pdtest→HPL_pdgesv			
A	HPL_T_pmat *	136.6(88.3)	2.9
main→HPL_pdtest→HPL_pdgesv→HPL_pdgesv0			
PANEL→L2	HPL_T_pmat	112.8(72.9)	8.5
PANEL→A	double	12.8(8.3)	3.8
PANEL→U	double	10.2(6.6)	5.2

Blame Points

Implementation Details

- Mixture of Static and Runtime Tools
- Static Analysis
  - LLVM
  - Boost
- Runtime Analysis
  - Dyninst API
  - Symtab API
  - Stackwalker API
  - PAPI

LLVM (Low Level Virtual Machine)

- What is it?
  - Compiler Infrastructure
  - Provides Intermediate Representation
    - Each instruction in SSA form
- Why we use it?
  - Need intermediate representation for static analysis
  - SSA form useful for creating dependency relationships
  - Intuitive API for accessing
    - Def-use chains
    - Dominator & CFG information
    - Language Independent representation of complex types
  - Integration with GCC
  - Multiple Language support
    - C, C++, Fortran
- Limitations
  - llvm-gcc versus gcc

Boost

- What is it?
  - Widely used portable C++ Libraries
- Why we use it?
  - Implicit/Explicit data flow relationships
    - Can create very large graphs
  - Boost provides graph libraries
    - Efficient representation of nodes/edges
      - Descriptors assigned to both
    - DFS, BFS, Uniform Cost Search
    - Dijkstra's Shortest Path, Kruskal's MST, …
- Limitations
  - Trade efficiency for requiring one more library

StackWalker API

- What is it?
  - API for runtime traversing of stack
- Why we use it?
  - Instance Generation
    - Used in combination with PAPI
    - Each sample point we need full path information
    - Use full context given from PAPI
      - Walk up stack until we reach the top
  - Mem-Container Information
    - Used in combination with Dyninst
    - Wrapper functions mean we need full path
      - Every allocation we get full allocation path
- Limitations
  - Frame pointer removal decreases accuracy

DyninstAPI

- What is it?
  - Dynamic instrumentation tool
- Why we use it?
  - Need to instrument memory allocation sites
  - Integrated with StackWalkerAPI
- Limitations
  - Instrumentation overhead

SymtabAPI

- What is it?
  - API for accessing symbol information
- Why we use it?
  - General Module/Function Information
  - Line Number Mappings
    - Runtime information mapped back to source
    - Use line number mappings for this
- Limitations
  - Debugging Information needed

PAPI

- What is it?
  - API that provides interface to hardware counters
- Why we use it?
  - Instance (Sample Point) Generation
    - PAPI provides sampling interface
    - User chooses metric to trigger sample
      - Metrics can be any measurable event on system
      - PAPI hardware counters
- Limitations
  - Special kernel patch required on certain systems

Advantage of Using Tools

Application/API	LOC (w/comments)
Blame	6K (8K)
Dyninst API 6.0	292K (360K)
Symtab API 6.0	51K (65K)
Stackwalker API 6.0	52K (66K)
LLVM 2.3	298K (375K)
PAPI 3.6	278K (320K)
Boost (Graph) 1.36	29K (33K)

Conclusion

- Variable "blame" mapping
  - Switch analysis from delimited regions to variables
  - Alternative to standard profiling techniques
- Lessons Learned
  - Standards are a good things
    - PAPI gives ucontext
    - Stackwalker uses information for context
  - Best not to reinvent the wheel … BUT
  - Tool interoperability can be a problem
    - Compiler, OS compatibilities
    - Runtime tool interoperability
    - Target application/end-user requirements
- Questions?